May 1982

Report. No. STAN-CS-82-906

Truncated-Newton Methods

by

Stephen Gregory Nash

DTIC QUALITY INSPECTED 2

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Department of Computer Science

Stanford University
Stanford, CA 94305

19970103 067

TRUNCATED-NEWTON METHODS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

by
Stephen Gregory Nash
May 1982

TRUNCATED-NEWTON METHODS

Stephen G. Nash, Ph.D.

Stanford University, 1982

The problem of minimizing a real-valued function F of n variables arises in many contexts. Most methods for solving this problem have their roots in Newton's method, i.e. they are based on approximating F by a quadratic function Q. If the number of variables n is large, then Newton's method can be problematic since it requires the computation and storage of the Hessian matrix of second derivatives. Use of finite-differencing and sparse-matrix techniques has overcome some of these problems but not all.

In this thesis, we examine a flexible class of methods, called truncated-Newton methods. They are based on approximately minimizing the quadratic function Q using an iterative scheme such as the linear conjugate-gradient algorithm. A truncated-Newton algorithm is made up of two sub-algorithms: an outer non-linear algorithm controlling the entire minimization, and an inner linear algorithm for approximately minimizing Q.

The most important choice is the selection of the inner algorithm. When the Hessian matrix is known to be positive-definite everywhere, then the basic linear conjugate-gradient algorithm can be used. If not, Q may not have a minimum. We have used the correspondence between the linear conjugate-gradient algorithm and the Lanczos algorithm for tridiagonalizing a symmetric matrix to develop methods for the indefinite case.

The performance of the inner algorithm can be greatly improved through the use of preconditioning strategies. Preconditionings can be developed using either the outer non-linear algorithm or using information computed during the inner algorithm. A number of diagonal and tridiagonal preconditioning strategies are derived here.

Numerical tests show that a carefully chosen truncated-Newton method can perform significantly better than the best non-linear conjugate-gradient algorithms available today. This is important since the two classes of methods have comparable storage and operation counts, and they are the only methods available for solving many large-scale problems.

Acknowledgments

I would like to thank the members of my reading committee—Walter Murray, Philip Gill, Gene Golub, and Robert Schreiber—for their advice and support in this work. The problem was originally suggested by Walter Murray, and he has continued to make insightful comments throughout the research. Philip Gill has also proved to be a valuable mentor.

The general atmosphere within the Numerical Analysis Group and in the Computer Science Department as a whole was very conducive. To a great extent, this is a result of Gene Golub's enthusiasm for the field.

The work in this dissertation was supported in part by the National Science Foundation Grant MCS 78-17697, by U.S. Army Research Office Grant DAAG29-78-G0179, and by a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship. It was prepared using Donald Knuth's TEX typesetting system. The facilities of the Stanford Linear Accelerator Center were used for the numerical computations.

Finally, I would like to thank my family and friends for their support over the past five years.

Table of Contents

1 Introduction

1.1. Motivation

The problem of minimizing a real-valued function of n variables

$$\min_x F(x) \tag{1.1.1}$$

arises in many contexts and applications. Over the years, a large variety of methods have been derived to solve this problem. Many of these methods have their roots in Newton's method, i.e. they are based on approximating F by a quadratic function using first- and second-derivative information at the current point. The quadratic function is then minimized, and it is hoped that this minimum gives information about the minimum of the original function.

Much work has been done to adapt and improve this basic method. In part the motivation for these changes is that the basic method is not always defined. For example, if the Hessian matrix is indefinite at some iteration, then the quadratic does not have a minimum.

Variations in the methods for problem (1.1.1) have also been derived for reasons based on the nature of the objective function F. There are basically two difficulties which can arise. Firstly, if the number of variables n is large, then storage limitations can make it difficult to store information about the problem. Secondly, the second derivatives of the function F may be very expensive (or impossible) to compute.

Because Newton's method in its traditional form requires the computation and storage of the $n \times n$ matrix of second derivatives, it can be problematic for both of these reasons. Use of finite-differencing and sparse-matrix techniques has overcome some of these problems, but not all.

The other chief classes of methods are Quasi-Newton and Conjugate-Gradient algorithms. Quasi-Newton methods do not require any second-derivative information; they still require the storage of an $n \times n$ matrix. Conjugate-gradient methods, however, remove even this difficulty, since they only require a few n vectors.

These difficulties are not overcome without some cost. In terms of the total number of iterations (or function evaluations) required to solve a minimization problem, Newton's method is extremely efficient. Quasi-Newton methods can often approach this efficiency

on small problems, but the performance of conjugate-gradient methods is by comparison erratic.

These differences in requirements and performance for the three classes of methods are unfortunate. They imply that all minimization problems must also be put into one of three classes, based on which algorithm is best capable of solving them on a given machine. Unfortunately, differences between problems are not always very sharp. It would be preferable if the distinctions between algorithms were not as great.

In the last few years, work has been done to fill in the gap between conjugate-gradient and quasi-Newton methods. This work comes under the category of limited-memory Quasi-Newton methods. More recently still, truncated-Newton methods have been developed. In the context in which we will develop them, they can be viewed as a synthesis of all three basic methods.

The great advantage of truncated-Newton methods is their flexibility. They can be adjusted to emulate any of the standard algorithms as well as everything in-between. They have variable storage requirements. It is possible to adjust them to use varying amounts of second-derivative information. It is also possible to fine-tune these methods to the needs of the problem being solved. Potentially, they can also adapt to changes in the behavior of the function being minimized.

In addition, we are concerned with the effect the computing environment has on the choice of an algorithm. When using a large, central computing facility complete with program libraries and technical consultants, then efficiency and stability of the method are the only considerations. However, when a small machine is the primary device available, then the size and complexity of the program must also be taken into account. This situation is becoming ever more important as the cost of small machines continues to drop, and as distributed computing becomes a more popular way of allocating machine resources.

The main topic of this thesis is the effective implementation of truncated-Newton methods. After some necessary preliminaries, it begins with a discussion of the three traditional classes of algorithms, along with a discussion of the techniques which are used to make them useful for larger classes of problems. This is followed by a description of truncated-Newton methods in their most basic form, along with a discussion of some of the underlying algorithms that might be used to implement them. Chapter 4 deals with

convergence criteria for the sub-algorithm, and Chapter 5 with preconditioning, which is essential if these methods are to be competitive. Chapter 6 discusses extensions to constrained and least-squares problems. Numerical results are presented in Chapter 7. Finally, an extensive discussion of how to adapt truncated-Newton methods for specific problems (both through a *priori* information and through dynamic modification of the algorithm) is given in Chapter 8.

1.2. Notation and Basic Theory

The principal problem we are trying to solve in this thesis is

$$\min_{x \in \Re^n} F(x), \tag{1.2.1}$$

where $F(x)$ is a nonlinear real-valued function of the variables

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

and $\Re^n$ denotes the n-dimensional Euclidean space. The gradient of F will be denoted by the vector g where

$$g_i = \frac{\partial F(x)}{\partial x_i}, \qquad i = 1, 2, \ldots, n,$$

and G will be used to denote the $n \times n$ matrix of second derivates, i.e.

$$G_{ij} = \frac{\partial^2 F(x)}{\partial x_i \partial x_j} \qquad \begin{matrix} i = 1, 2, \ldots, n \\ j = 1, 2, \ldots, n. \end{matrix}$$

All methods considered here for solving (1.2.1) will be descent methods; that is, the value of the objective function $F(x)$ will be decreased at each iteration. More specifically, except in the section which describes trust-region methods, we will principally be concerned with line-search algorithms. As a result, all of our algorithms will have the following general form:

1.2.2. Descent Algorithm

D1. Given $x^{(k)}$, the k^{th} approximation to x^*, a minimum of $F(x)$.

D2. Compute $p^{(k)}$, a direction of search, such that ${p^{(k)}}^T g^{(k)} < 0$.

D3. Find $\alpha^{(k)} > 0$, a scalar step-length, such that $F(x^{(k)} + \alpha^{(k)} p^{(k)}) < F(x^{(k)})$.

D4. Set $x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)} p^{(k)}$ and return to step **D2**.

Step **D3** is called the line-search, and it will be discussed later in this chapter. For our purposes, step **D2** will be the most significant, for the process used to compute $p^{(k)}$ generally classifies the entire algorithm. This computation is often based on the gradient or the Hessian at the point $x^{(k)}$ (denoted by $g^{(k)}$ and $G^{(k)}$, respectively), or on information accumulated in previous iterations.

A considerable amount of the work in step **D2** is dependent on results from linear algebra. The principal theoretical results will be presented in the next section, but first some notational details will be discussed here.

In general, matrices will be denoted by upper-case Roman letters (G), and their elements will be specified using subscripts (G_{ij}). Vectors will be denoted by lower-case Roman letters, with subscripts again being used for individual elements (g, g_i). Scalars will be denoted by lower-case Greek letters (α). A superfix ^{T} on a matrix or vector denotes transpose. $\|y\|$ denotes the Euclidean norm of the vector y. Other than those vectors already mentioned, in the context of optimization there are two additional vectors which have special meaning. These are

$$s^{(k)} \equiv x^{(k+1)} - x^{(k)},$$

the difference between the successive estimates of the minimum, and

$$y^{(k)} \equiv g^{(k+1)} - g^{(k)},$$

the difference between the successive gradient values.

In order to be able to terminate algorithm (1.2.2), it is important to know how to identify x^*, the point which minimizes the function F. The following theorem gives necessary and sufficient conditions for the minimum of an unconstrained real-valued function.

Theorem 1.2.3 (a) Let x^* be a relative minimum point of the twice continuously differentiable function F. Then $g(x^*) = 0$ and $G(x^*)$ is positive semi-definite. (See section 1.3 for a definition of positive semi-definite.)

(b) Suppose that F is a twice continuously differentiable function mapping from $\Re^n$ to $\Re$. Suppose in addition that x^* is a point in $\Re^n$ for which $g(x^*) = 0$ and $G(x^*)$ is positive definite. Then x^* is a strict relative minimum point of F.

1.3. Basic Results in Linear Algebra

The information in this section will be presented briefly and without proof. A much more complete discussion can be found in Wilkinson [1965], Chapter 1.

Definitions:

1. A matrix A is *symmetric* if $A = A^T$.
2. A symmetric matrix A is *positive definite* if

$$y^T A y > 0, \qquad \forall y \neq 0.$$

3. A symmetric matrix A is *positive semi-definite* if

$$y^T A y \geq 0, \qquad \forall y.$$

[Similar definitions exist for *negative definite* and *negative semi-definite* matrices. A matrix falling into none of these categories is called *indefinite*.]

4. A set of vectors $\{a_1, \ldots, a_n\}$ is *linearly independent* if

$$\sum_{j=1}^{n} \beta_j a_j = 0$$

implies that

$$\beta_j = 0, \qquad j = 1, \ldots, n.$$

5. The *space spanned* by a set of vectors is the space generated by all linear combinations of those vectors.
6. The *rank* of a matrix A is equal to the maximum number of linearly-independent rows.
7. The *range* of a matrix A, denoted by $R(A)$, is the space spanned by the columns of A.
8. The *null space* of a matrix A, denoted by $N(A)$, is $\{x \mid A^T x = 0\}$.
9. The *condition number* of a non-singular matrix A is defined to be

$$\kappa(A) \equiv \|A\| \cdot \|A^{-1}\|,$$

where $\|\cdot\|$ is the 2-norm of a matrix.

10. A matrix A is *lower (upper) triangular* if

$$A_{ij} = 0, \qquad i < j \quad (i > j).$$

A is *unit lower (upper) triangular* if, in addition, $A_{ii} = 1, \quad \forall i.$

Results:

1. Let A be an n by n symmetric matrix. Then there exist n orthonormal vectors $v_1, \ldots, v_n$ and n scalars $\lambda_1, \ldots, \lambda_n$ such that

$$Av_i = \lambda_i v_i, \qquad i = 1, \ldots, n.$$

The vector v_i is an *eigenvector* of A, and λ_i is its associated *eigenvalue*.

2. A symmetric matrix of rank r has r non-zero eigenvalues.

3. A positive-definite matrix has positive eigenvalues.

4. A symmetric matrix A is positive definite if and only if it can be factored as

$$A = LDL^T,$$

where L is unit lower triangular and D is diagonal with positive diagonal entries. [*Cholesky factorization*]

1.4. Line Search Techniques

As in the previous section, this will only be a brief discussion of the topic of line searches. More complete information can be found in Gill and Murray [1979] and [1974b].

Step **D3** of algorithm (1.2.2) requires that a scalar α be found such that

$$F(x + \alpha p) < F(x). \tag{1.4.1}$$

[The superscript $^{(k)}$ will be dropped for reasons of clarity.] One way to achieve this is to require that

$$F(x + \alpha p) = \min_{\tilde{\alpha} > 0} F(x + \tilde{\alpha} p). \tag{1.4.2}$$

Although necessary for 1-dimensional minimization, this condition is overly stringent in a higher-dimensional context (and of little use for constrained optimization).

At the opposite extreme, it is not sufficient to choose just any value of α such that

(1.4.1) is satisfied. To see this, consider the simple 1-dimensional problem:

$$\begin{aligned} F(x) &= x^2, \\ x^{(1)} &= 2, \\ p^{(k)} &= -1, \qquad \forall k, \\ \alpha^{(k)} &= 2^{-k}. \end{aligned}$$

It is easily seen that the sequence $\{x^{(k)}\}$ satisfies (1.4.1) but that

$$\lim_{k \to \infty} x^{(k)} = 1 \neq 0 = x^*.$$

In order to efficiently minimize functions of several variables, and also to be able to guarantee convergence of optimization algorithms, a compromise between these two positions has been found to be effective. In this regard, two concepts have been shown to be of considerable value. The first is that the search direction must not become arbitrarily close to being orthogonal to the steepest descent direction (that is, the gradient). Usually this is achieved by the method used to choose the search direction.

If this property is satisfied, then the second condition is that the function $F(x)$ must be "sufficiently" reduced at each iteration. This condition is often achieved by an appropriate choice of step length within a line search algorithm. One such algorithm is presented here.

(1.4.3) Line search algorithm

Let $\{\alpha_j\}$ define a sequence of points that tend in the limit to the minimum of $F(x)$ along p. (If $F(x)$ is smooth, this sequence can be computed by means of some safeguarded polynomial interpolation algorithm.) Let t be the index of the first member of this sequence such that

$$|g(x + \alpha_t p)^T p| \leq -\eta g^T p \tag{1.4.4}$$

where η $(0 \leq \eta < 1)$ is some constant scalar. Let μ $(0 < \mu \leq \frac{1}{2})$ be another constant scalar. Find the smallest non-negative integer r such that

$$F(x) - F(x + 2^{-r}\alpha_t p) \geq -2^{-r}\alpha_t \mu g^T p \tag{1.4.5}$$

and set $\alpha = 2^{-r}\alpha_t$.∎

If α is computed according to this rule, it can be shown (Gill and Murray [1973a])

that

$$F(x) - F(x + \alpha p) > \phi\left(\frac{-g^T p}{\|p\|}\right),$$

where ϕ is a real-valued function such that, for any sequence $\{c_k\}$,

$$\lim_{k\to\infty} \phi(c_k) = 0 \qquad \text{implies} \qquad \lim_{k\to\infty} c_k = 0.$$

An important property of conditions (1.4.4) and (1.4.5) is that if μ is chosen as a small value (say 10^{-4}) then, unless $F(x)$ is a pathologically ill-behaved function, any value of α_t satisfying (1.4.4) automatically satisfies (1.4.5) with $r = 0$. In this case the line search algorithm reduces to finding a scalar α such that

$$|g(x + \alpha p)^T p| \leq -\eta g^T p.$$

The value of η can be specified by the user and can be used to give a step length that is well-suited to the problem being solved. If η is chosen as 0.9, the algorithm will generally compute a "crude" value of α, provided it satisfies (1.4.5). This value will often be α_0, the initial guess for α. If η is chosen as zero, α will satisfy (1.4.2), the condition for an exact line search.

In order for certain asymptotic convergence rates to be attained, it is often necessary that ultimately $\alpha = 1$ for $\forall k > K$. For this reason, $\alpha_0 = 1$ is a common feature of line-search algorithms, but it is certainly not the only possibility. Davidon [1959] suggested the following choice for α_0

$$\alpha_0 = -2(F^{(k)} - F^{(\text{est})})/g^{(k)^T} p^{(k)}$$

whenever the quantity on the right hand side is less than or equal to 1. Here, $F^{(\text{est})}$ is a user-specified estimate of the function value at the solution. (It is common for the user to have some *a priori* information about the function F. If not, the choice $\alpha_0 = 1$ can be used.) This formula has been found to be quite successful computationally.

For nonlinear conjugate-gradient algorithms (described in the next chapter), a further condition in the line search is required in order to insure the descent property of the search direction $p^{(k+1)}$ at the next iteration. Details of these requirements can be found in Gill and Murray [1979].

One final remark that is relevant to the general topic of this thesis concerns the computation of the sequence $\{\alpha_j\}$ in (1.4.4). In many situations, this sequence will be

computed using function and gradient values. However, when the gradient and function are expensive to compute relative to the cost of computing the function alone (i.e. twice the cost), this sequence can be computed using function values only (see Gill and Murray [1974b]).

1.5. Rates of Convergence

In the chapters to follow, we will be concerned with the speed at which various algorithms converge. The following definitions will be useful for our purposes.

1. A sequence $\{x^{(k)}\}$, converging to x^*, is said to converge with Q-order m if

$$\lim_{k\to\infty} \frac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|^m} < \infty. \tag{1.5.1}$$

2. The sequence $\{x^{(k)}\}$ is said to converge Q-superlinearly if

$$\lim_{k\to\infty} \frac{\|x^{(k+1)} - x^*\|}{\|x^{(k)} - x^*\|} = 0. \tag{1.5.2}$$

3. The sequence $\{x^{(k)}\}$ is said to converge with R-order m if

$$\|x^{(k)} - x^*\| \leq \beta_k \qquad k = 0, 1, \ldots \tag{1.5.3}$$

where $\{\beta_k\}$ is a sequence that converges to zero with Q-order m.

The most important instances of definition 1 are $m = 1$ (Q-linear convergence) and $m = 2$ (Q-quadratic convergence). Because Q-order rates of convergence are more important for our purposes, the label Q- will often be dropped in the discussions to follow. Only R-order rates will be distinguished.

For a more complete treatment of rates of convergence, see Ortega and Rheinboldt [1970], pp. 281–298.

2 The Basic Methods

2.1 Introduction

Although we have so far mentioned only three basic classes of methods for solving the unconstrained minimization problem (1.2.1), within each class there are a great many varieties, and choice of the exact algorithm to use is not always easy. In the absence of other considerations, Newton's method is almost always the method of choice, at least in its modern safeguarded versions. Other methods can be considered as compromises to Newton's method.

The three classes of methods will be discussed from that perspective, since it leads naturally to the discussion of truncated-Newton methods in the next chapter. Newton's method will be discussed first; this will be followed by descriptions of quasi-Newton and conjugate-gradient methods. In the last section, extensions and adaptations of these algorithms to larger classes of problems are presented.

It should be pointed out that all the methods in this and the next chapter will be developed as linesearch algorithms. Alternative versions of these methods employing trust-region schemes will be presented in a later chapter.

2.2 Newton's Method

Since it is impossible to have a direct method for solving the unconstrained minimization problem (1.2.1) in general (such a direct method would imply that there existed a direct method for finding roots of arbitrary polynomials), we must rely on iterative methods. In the case of minimization, these usually take the form of finding some approximation to the objective function F, computing its minimum, and then using this point to compute a better approximation to the minimum of the original function.

For Newton's methods, this approximate function is based on the expansion of F in a Taylor series:

$$F(x^{(0)} + p) = F(x^{(0)}) + p^T \nabla F(x^{(0)}) + \tfrac{1}{2} p^T \nabla^2 F(x^{(0)})\, p + \mathrm{R}_3(x^{(0)}, p), \qquad (2.2.1)$$

where $\mathrm{R}_3(x^{(0)}, p)$ represents the higher-order terms in the series. From now on, we will denote the gradient of F as

$$g = g(x) \equiv \nabla F(x),$$

and the Hessian as

$$G = G(x) \equiv \nabla^2 F(x).$$

To derive Newton's method, we drop the remainder term in (2.2.1). If we denote $F^{(0)} \equiv F(x^{(0)})$, then

$$\begin{aligned} \min_x F(x) &= \min_p F(x^{(0)} + p) \\ &\approx \min_p Q(p) \\ &\equiv \min_p \left[F^{(0)} + g^T p + \tfrac{1}{2} p^T G p \right] \end{aligned} \tag{2.2.2}$$

We have now approximated F by a quadratic function. Taking the derivative of $Q(p)$ in (2.2.2) and setting it equal to zero, we obtain the so-called Newton equations for the step to the minimum of the quadratic function:

$$Gp = -g. \tag{2.2.3}$$

This formula, and its equivalent version in (2.2.2) will be fundamental in the work to follow.

In its basic form, Newton's method cannot be used, since it is not always defined or meaningful. For example, away from the solution x^*, there is no guarantee that the matrix G will be positive definite. This means that (2.2.3) may not have a solution, or that the minimum in (2.2.2) may not exist. Also, we are insisting that the search direction p be a descent direction, i.e. $p^T g < 0$, which may not be true if G is not positive definite.

Many authors have suggested ways of safe-guarding Newton's method so that the search-direction will be appropriately defined at each iteration. The most successful of these schemes involve replacing G by a related positive-definite matrix (see Murray [1972]).

Indefiniteness is usually detected and corrected by computing and, if necessary, adjusting some decomposition of the matrix G. Greenstadt [1967] proposed using a spectral decomposition and replacing negative eigenvalues with their moduli. Unfortunately, this is a very expensive operation—and frequently unnecessary as G is often positive definite. Indefiniteness can also be detected using a Cholesky factorization (or one of its variants). Details of this work can be found in Gill and Murray [1974a], Bunch and Parlett [1971], Dax and Kaniel [1977], etc.

It is not enough to replace G by *any* positive-definite matrix $\overline{G}$. If the norm of $E \equiv \overline{G} - G$ is large, then $\|p\|$ will be small, and the algorithm may not converge. It is

thus important to be able to bound the norm of the modification matrix E.

Simply replacing G by a positive-definite matrix is not sufficient to define a method. Other problems can also occur during a minimization process. For example, it is not always clear how to obtain a descent direction at a saddle point, that is, when $g = 0$ and G is not positive definite. The methods described in the previous paragraph can all be used to compute a direction of negative curvature in this event. Such a direction is defined by the condition

$$p^T G p < 0.$$

An advantage of the Gill and Murray approach is its simplicity, and there is no evidence that it is less efficient than alternative methods. Since the truncated-Newton algorithm in the next chapter owes much to this method, some details of the algorithm will be discussed briefly here.

This algorithm is based on the result that a symmetric matrix G is positive definite if and only if it can be factored in the form

$$G = LDL^T, \tag{2.2.4}$$

where D is a diagonal matrix with positive diagonal entries and L is a unit lower triangular matrix. This suggests the following technique. Attempt to compute the Cholesky factorization (2.2.4) of G. If at any stage D_{ii} turns out to be negative or zero, add some positive quantity E_{ii} to G_{ii} to correct the problem. Monitor the size of the elements of L, and if they are "too large," further increase the size of E_{ii}. Now continue with the factorization.

Exact formulas for this process can be found in the Gill and Murray paper. The end result is that we have computed the Cholesky factorization of

$$G + E = LDL^T, \tag{2.2.5}$$

where $E = \text{diag}(E_{ii})$. It can be shown that E is identically zero whenever G is "sufficiently positive-definite." Since it would probably be necessary in any case to compute the factorization (2.2.4) to solve the Newton equations (2.2.3), the factorization (2.2.5) can be considered as a natural by-product of the basic Newton method. Using this factorization, we obtain the modified-Newton equations for the step to the minimum of the modified quadratic function:

$$(G + E)p = -g. \tag{2.2.6}$$

The resulting modified-Newton algorithm can be described as follows:

(2.2.6) Modified-Newton Algorithm

N1. Given x.

N2. Compute p from (2.2.3), (2.2.6), or related formulas.

N3. Find α such that $F(x + \alpha p) < F(x)$.

N4. Set $x \leftarrow x + \alpha p$ and return to step **N2.**

This algorithm has been left deliberately vague in order to allow for the many possibilities discussed above for steps **N2** and **N3.**

Under a variety of conditions, it can be shown that the above algorithm is globally and locally quadratically convergent. Although more general results can be found (see, for example, Ortega and Rheinboldt [1970], pp. 421–430), the following theorem will be adequate for our purposes.

Theorem (2.2.7) Suppose that $F : \Re^n \to \Re$ is twice continuously differentiable in an open convex set D and that there is an x^* in D such that $g(x^*) = 0$ and $G(x^*)$ is positive definite. Then there is an open set S which contains x^* such that for any $x^{(0)} \in S$ the Newton iterates are well-defined, remain in S and converge to x^*. Moreover, there is a constant β such that

$$\|x^{(k+1)} - x^*\| \leq \beta \|x^{(k)} - x^*\|^2, \qquad k = 0, 1, \ldots.$$

2.3. Quasi-Newton Methods

The methods of this section have also been referred to in the literature as secant methods, variable-metric methods, etc. Here, we will refer to them as quasi-Newton methods.

In the previous section we computed the search direction $p^{(k)}$ as the minimum of a strictly convex quadratic function or, equivalently, as the solution of the system of linear equations (2.2.6). Both of these methods require the Hessian matrix $G^{(k)}$. It might be hoped that similarly successful methods might be derived from minimizing

$$Q(p) = \tfrac{1}{2} p^T B^{(k)} p + p^T g^{(k)}, \tag{2.3.1}$$

or equivalently, solving

$$B^{(k)} p = -g^{(k)}, \tag{2.3.2}$$

where $B^{(k)}$ is some suitably chosen approximation to $G^{(k)}$.

In one dimension, there is a well-known algorithm of this type, called the secant method. In that method,

$$B^{(k+1)} = \frac{g^{(k+1)} - g^{(k)}}{x^{(k+1)} - x^{(k)}} = \frac{y^{(k)}}{s^{(k)}}.$$

Multiplying through by $s^{(k)}$ gives

$$B^{(k+1)} s^{(k)} = y^{(k)}. \tag{2.3.3}$$

Considering the methods derived from (2.3.2) as extensions of the one-dimensional secant method, it is natural to insist that (2.3.3) be satisfied in the n-dimensional case as well. In this context, (2.3.3) will be referred to as the *quasi-Newton condition* and will be usd to define this class of methods.

In n dimensions, (2.3.3) is insufficient to uniquely specify $B^{(k+1)}$ and further restrictions are required to insure that an algorithm of this type is well-defined. Although the historical development of these methods was somewhat haphazard, it is now known that all the major quasi-Newton methods can be derived in the following fashion:

(2.3.4) Quasi-Newton Update

U1. Given $B^{(k)}$.

U2. Choose a set of properties P which $B^{(k+1)}$ must preserve. (Some typical choices are symmetry, positive-definiteness, a particular sparsity pattern, etc.)

U3. Choose a matrix norm $\|\cdot\|_M$.

U4. Define $B^{(k+1)}$ as the matrix which satisfies (2.3.3), preserves the properties P, and which minimizes $\|B^{(k+1)} - B^{(k)}\|_M$.

Various specific updates are then obtained by specifying the set of properties P and by choosing a norm $\|\cdot\|_M$. The following few paragraphs will outline the principal updates in use today. [For simplicity in the following discussion, we will denote $B \equiv B^{(k)}$, $\overline{B} \equiv B^{(k+1)}$.]

A simple formula, known as Broyden's update, is obtained by letting P be void and by choosing $\|\cdot\|_M$ to be the Frobenius norm. Then

$$\overline{B} = B + \frac{(y - Bs)s^T}{s^T s}. \tag{2.3.5}$$

For solving systems of non-linear equations, (2.3.5) is quite satisfactory, but for function minimization it is inadequate. For example, because the Hessian matrix is symmetric, it would be desirable to insist that the update formula have the property of hereditary symmetry; that is, that $\overline{B}$ be symmetric whenever B is.

Two choices for $\|\cdot\|_M$ lead to the two following symmetric updates. If $\|\cdot\|_M$ is chosen as the Euclidean 2-norm, then the symmetric rank-one formula is obtained:

$$\overline{B} = B + \frac{(y - Bs)(y - Bs)^T}{(y - Bs)^T s}. \tag{2.3.6}$$

On the other hand, if $\|\cdot\|_M$ is chosen to be the Frobenius norm, then the Powell Symmetric Broyden (PSB) update is obtained:

$$\overline{B}_{\text{PSB}} = B + \frac{(y - Bs)s^T + s(y - Bs)^T}{s^T s} - \frac{(y - Bs)^T s}{(s^T s)^2} ss^T. \tag{2.3.7}$$

Another property of considerable significance is hereditary positive-definiteness. If the matrix B is guaranteed to be positive definite and bounded, then the search direction is guaranteed to be a descent direction. If a positive-definite approximation is not chosen, then modification schemes similar to those necessary for Newton's method would need to be invoked. Since G is positive semi-definite at the solution, the restriction that B be positive definite will not asymptotically prevent superlinear convergence.

The results for positive-definite updates are not quite as simple as for symmetric updates. Again, choosing $\|\cdot\|_M$ to be the 2-norm or the Frobenius norm leads to the following two update formulas. But in this situation, the vectors y and s are no longer arbitrary. For the updates to retain positive-definiteness, it is necessary that $y^T s > 0$. This property can be assured to hold by performing a sufficiently accurate line search. With this in mind, then, the two updates are, with the 2-norm (the Davidon-Fletcher-Powell (DFP) update):

$$\overline{B}_{\text{DFP}} = B + \frac{(y - Bs)y^T + y(y - Bs)^T}{y^T s} - \frac{(y - Bs)^T s}{(y^T s)^2} yy^T \tag{2.3.8}$$

and with the Frobenius norm (the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update):

$$\overline{B}_{\text{BFGS}} = B + \frac{yy^T}{y^T s} - \frac{(Bs)(Bs)^T}{s^T Bs}. \tag{2.3.9}$$

Computationally, it has been found that (2.3.9) is the most successful update known for general minimization (see Gill, Murray, and Pitfield [1972]).

A quasi-Newton approximation can be initialized in a number of ways. The simplest choice, and the only choice in the absence of addition information about the Hessian, is to set $B^{(0)} = I$. If it is feasible to compute some second-derivative information, this can also be used to initialize the approximation, as can information from previous minimization steps, if the problem being solved is one of a sequence of similar problems. If a trust-region strategy is being used in combination with the quasi-Newton method, then the bounds on the variables can be used to derive an initial approximation to the Hessian (see Powell [1970]).

To compute the search direction, it is necessary to solve the system of linear equations (2.3.2). It may appear that this will require $O(n^3)$ operations at each iteration, but actually, it is possible to reduce this to $O(n^2)$ operations in either of two ways.

The first observation is that since all of the update formulas are of low rank, it is simple to apply the Sherman-Morisson formula and obtain low-rank updates for $H \equiv B^{-1}$ (see Stewart [1967]). Using the inverse update, the solution of (2.3.2) would amount to no more than multiplication by the matrix H, an $O(n^2)$ process. However, this is unstable.

The second idea, and this is what is used in the best algorithms today, is to update a factorization of B rather than B itself. For example, if a symmetric approximation B is stored in the form

$$B = LL^T,$$

then solution of (2.3.2) involves only two back-substitutions, again an $O(n^2)$ process. Details of how various matrix factorizations can be updated efficiently can be found in Gill, Golub, Murray and Saunders [1974].

Under fairly mild restrictions, quasi-Newton methods can generally be shown to exhibit global and superlinear convergence. The following theorem (from Dennis and Moré [1977], page 82) is typical and adequate for our purposes.

Theorem 2.3.1. Let $F : \Re^n \rightarrow \Re$ be twice continuously differentiable in an open convex set D, and assume that $g(x^*) = 0$ and that $G(x^*)$ is positive definite for some x^* in D. Suppose in addition that

$$\|G(x) - G(x^*)\| \leq \gamma \|x - x^*\|$$

for some constant γ and for all x in D. Suppose that the algorithm (1.2.2) is implemented

by choosing

$$B^{(k)}p^{(k)} = -g^{(k)}$$

where $B^{(k)}$ is obtained using a BFGS or DFP update with $B^{(0)} = I$. Also, suppose that $\alpha^{(k)}$ is determined using the line search formulas (1.4.4) and (1.4.5), and that

$$\sum_{k=0}^{\infty} \|x^{(k)} - x^*\| < +\infty.$$

Then $\{x^{(k)}\}$ converges superlinearly to x^*.∎

2.4. Nonlinear Conjugate-gradient Algorithms

The (linear) conjugate-gradient algorithm of Hestenes and Stiefel [1952] was originally designed to solve the system of linear equations

$$Ax = b \tag{2.4.1}$$

where A is a positive-definite square matrix. As indicated earlier, this is equivalent to minimizing the quadratic form

$$Q(x) = \tfrac{1}{2}x^TAx - x^Tb. \tag{2.4.2}$$

For clarity, we will give an outline of the algorithm here. A more complete derivation from another point of view can be found in section 3.2.

The solution x of (2.4.1) will be computed as a linear combination of A-conjugate directions. That is,

$$x = \sum_i \alpha_i p_i \tag{2.4.3}$$

where

$$p_i^TAp_j = 0, \quad \text{for} \quad i \neq j.$$

Notice that the concept of A-conjugacy is equivalent to the concept of orthogonality if the inner-product is defined as

$$(p_1, p_2) \equiv p_1^TAp_2.$$

This implies that any set of A-conjugate vectors will also be linearly independent.

If the representation (2.4.3) for x is substituted into (2.4.2), then minimizing $Q(x)$

becomes equivalent to solving a sequence of one-dimensional minimization problems:

$$\begin{aligned}\min_x Q(x) &= \min_a Q\Big(\sum \alpha_k p_k\Big) \\ &= \min_a \left\{ \tfrac{1}{2}\Big(\sum \alpha_i p_i\Big)^T A\Big(\sum \alpha_j p_j\Big) - \Big(\sum \alpha_i p_i\Big)^T b \right\} \\ &= \min_a \left\{ \tfrac{1}{2}\sum_{i,j} \alpha_i \alpha_j p_i^T A p_j - \sum_i \alpha_i p_i^T b \right\} \\ &= \sum_i \left(\min_{\alpha_i} \left\{ \tfrac{1}{2}\alpha_i^2 p_i^T A p_i - \alpha_i p_i^T b \right\} \right).\end{aligned}$$

(due to the A-conjugacy of the vectors p_i). Each term in the final summation is a minimization problem involving only the coefficient α_i, so that the original minimization problem (2.4.2) has been completely decoupled into a set of trivial one-dimensional minimization problems.

The linear conjugate-gradient algorithm is quite simple to describe. Let x_0 be some initial guess of the solution to (2.4.1). At each stage, compute the current residual $r_k = b - Ax_k$. If $r_k = 0$, accept x_k as the solution of (2.4.1) and terminate the iteration. Otherwise, compute a new A-conjugate direction p_k (using the current residual and the previous A-conjugate direction p_{k-1}), and minimize $Q(x)$ along the line which starts at x_k and moves in the direction p_k (this corresponds to minimizing one term in the summation above).

Although the conjugate-gradient algorithm is described as an iterative method, it will terminate after a finite number of iterations in exact arithmetic. If m is the dimension of the system of equations, then there can be at most m A-conjugate vectors $\{p_k\}$ (because A-conjugate vectors must also be linearly independent). Since the solution x to (2.4.1) is expressed in terms of $\{p_k\}$, the conjugate-gradient algorithm must converge in at most m iterations.

One computational form of the linear conjugate-gradient algorithm is as follows:

Given an initial guess x_0.

For $k = 0, 1, \ldots$

$$r_k = b - Ax_k$$

$$\beta_k = r_k^T r_k / r_{k-1}^T r_{k-1}, \qquad \beta_0 = 0$$
$$p_k = r_k + \beta_k p_{k-1}$$

$$\alpha_k = r_k^T r_k / p_k^T A p_k$$
$$x_{k+1} = x_k + \alpha_k p_k.$$

A useful property of this algorithm is that the vectors $\{r_k\}$ are mutually orthogonal:

$$r_i^T r_j = 0, \qquad \text{if} \quad i \neq j,$$

and the vectors $\{p_k\}$ are mutually A-conjugate:

$$p_i^T A p_j = 0, \qquad \text{if} \quad i \neq j.$$

It is a relatively simple problem to adapt the linear conjugate-gradient algorithm to solve general nonlinear problems. Nonlinear conjugate-gradient algorithms are based on the Newton formulas (2.2.2) and (2.2.3). In this case, the underlying assumption is that $F(x)$ is a quadratic function, i.e. a function with a constant Hessian matrix. The formulas for the linear conjugate-gradient algorithm are then applied to the nonlinear function.

The nonlinear conjugate-gradient algorithm computes a new search direction using the formula

$$p^{(k+1)} = -g^{(k+1)} + \beta^{(k)} p^{(k)}. \tag{2.4.4}$$

The determination of $\alpha^{(k)}$ is now a univariate minimization problem. Various choices for the scalar $\beta^{(k)}$ lead to the various versions of the algorithm. The three principle formulas (all equivalent in the case where F really is a quadratic function) are:

1. Fletcher-Reeves

$$\beta^{(k)} = \|g^{(k+1)}\|_2^2 / \|g^{(k)}\|_2^2 \tag{2.4.5}$$

2. Hestenes-Stiefel

$$\beta^{(k)} = y^{(k)^T} g^{(k+1)} / y^{(k)^T} p^{(k)} \tag{2.4.6}$$

3. Polak-Ribiére

$$\beta^{(k)} = y^{(k)^T} g^{(k+1)} / \|g^{(k)}\|_2^2. \qquad (2.4.7)$$

In the non-quadratic case, these formulas have quite distinct properties. For example, (2.4.6) guarantees that $y^{(k)^T} p^{(k+1)} = 0$ (a property of the quadratic case) irrespective of the accuracy of the line search or any possible non-quadratic behavior of the objective function. Also, Powell [1977] has shown that (2.4.5) may lead to slow convergence in the general nonlinear case when exact line searches are performed.

In the quadratic case, the conjugate-gradient algorithm will theoretically converge in n iterations and will have generated n linearly independent search directions $p^{(k)}$. For this reason, Fletcher and Reeves [1964] suggested abandoning (2.4.4) after a cycle of n line searches, and setting $p^{(k+1)}$ as the steepest descent direction $g^{(k+1)}$. This strategy is known as *restarting*. Since then, Powell [1977] and others have suggested other restarting strategies, and these have considerably improved the performance of nonlinear conjugate-gradient algorithms.

McCormick and Pearson [1969] have shown that, for a wide class of functions, the restarted conjugate-gradient algorithm is n-step superlinearly convergent, i.e.

$$\lim_{k \to \infty} \frac{\|x^{(nk+n)} - x^*\|}{\|x^{(nk)} - x^*\|} = 0. \qquad (2.4.8)$$

This result depends critically on the use of restarting. Powell [1976] has shown that algorithms that do not contain a restarting strategy almost always converge linearly.

The result (2.4.8) may be somewhat misleading. In general, a successful conjugate-gradient algorithm should converge in $2n$ to $3n$ iterations, so that asymptotic behavior of the sort described by (2.4.8) would never be observed. Thus, for practical purposes, nonlinear conjugate-gradient algorithms can be considered to exhibit a linear rate of converge, and the motivation for restarting is not to achieve a superlinear rate of convergence.

2.5. Adaptations and Extensions of the Traditional Methods

The three main classes of methods are each associated with a particular class of problems to which they are ideally suited. The correspondences are as follows:

1. Newton's method: small and moderately sized problems where the

Hessian matrix is easy or cheap to compute.

2. Quasi-Newton methods: small and moderately sized problems where the Hessian is difficult or expensive to compute.

3. Conjugate-gradient methods: large problems.

For small problems, whenever possible a user would prefer to use Newton's method over quasi-Newton methods, and quasi-Newton methods over conjugate-gradient methods, because the relative efficiencies of the former methods are so much better. It should be remembered, however, that for particular problems we cannot be assured that a given method works better than another. There are reasons to suppose that the relative efficiencies of the three approaches are different for large problems, when it is possible to still apply all three methods (see Thapa [1980], Gill and Murray [1979]).

Much work has been done to extend Newton and quasi-Newton methods to larger classes of problems, and to modify conjugate-gradient methods to give them some of the properties of the other two classes of methods.

Newton's method is relatively easy to extend to large problems. At each iteration, it is necessary to solve a system of linear equations and, provided the Hessian is sparse, this can be achieved using sparse matrix methods (see, for example, Bunch and Rose [1976]). For most problems, however, second derivative information will not be available. By using a finite-difference approximation to G, Newton's method can be extended to the case where only first derivatives are known. In the dense case, this implies at least n additional gradients would be required. For large problems or even moderate-size problems, this is a prohibitive cost. Fortunately, if the Hessian is sparse it can be approximated in considerably less than n gradient evaluations. Details of this work can be found in Thapa [1980], Powell and Toint [1979].

A second possibility is to hold G fixed for several iterations. This idea has been explored by Brent [1973], but has been found to be less effective than using quasi-Newton methods (see, for example, Broyden [1971]). For special problems this method can work well. When the cost of factoring the Hessian matrix dominates the cost of computing it, or when the Hessian matrix is nearly constant, this can be the method of choice.

The extension of quasi-Newton methods to large problems is much more complex. The updates given in section 2.3 will not in general give a sparse Hessian approximation even if the actual Hessian is sparse. To overcome this deficiency, sparse updates have

been developed. This involves adding a sparsity condition in step U2 of the update algorithm (2.3.4). Schubert [1970] developed a sparse update for nonlinear systems of equations. A useful update for optimization was derived independently by Marwil [1978] and Toint[1978]. With these updates available, sparse matrix techniques are then used to solve the system of equations (2.3.2) for the search direction p.

Currently, it appears that sparse quasi-Newton methods are chiefly of theoretical interest. First of all, the sparse updates are no longer of low rank (in fact, they are in general of rank n) so that the converse of the dense case is now true and quasi-Newton methods take more algebraic operations per iteration than sparse Newton methods. Secondly, although they can be shown to converge theoretically at a superlinear rate, the asymptotic regime is in practice slow to set in, and these methods are usually less efficient than sparse-Newton methods (see Thapa [1980]).

Steihaug [1980] has derived a class of sparse approximate quasi-Newton updates, and has shown that algorithms based on them can be made to converge globally and superlinearly. This class is obtained by computing a sparse quasi-Newton update iteratively using the linear conjugate-gradient algorithm. No practical experience has yet been reported on these methods.

Although both Newton and quasi-Newton methods will continue to be improved, they critically depend on the assumption that the Hessian matrix is sparse. For unconstrained problems, this is almost always true. For constrained problems, however, the equivalent equations that require solution involve the projection of the Hessian matrix. Although the Hessian matrix and the matrices defining the projection may be sparse, the projected Hessian is often dense. Except in special cases, for such problems the direct application of Newton or quasi-Newton methods (as we have described them so far) is unlikely to be successful.

Because conjugate-gradient methods are so frugal in their storage requirements, attempts have been made to modify them to make them behave more like quasi-Newton methods. This group of methods is generally referred to as *limited-memory quasi-Newton methods.* Since the standard quasi-Newton updates described in section 2.3 only involve a few low-rank updates to an initial matrix, it is possible to apply them to a vector and only store the few vectors needed to define the low-rank portion of the update. The storage available determines the number of updates used. Because the size of the problem

prevents the solution of the system of equations (2.3.2), inverse updates are generally used in this application. Details of these algorithms, as well as a great many numerical examples illustrating their performance, can be found in Gill and Murray [1979].

Another technique which has been used to improve the performance of nonlinear conjugate-gradient algorithms is preconditioning. These algorithms will converge in one iteration if $f(x)$ is a quadratic function with Hessian matrix equal to the identity. The idea behind preconditioning is to use information about the problem (obtained either a *priori*, or dynamically as the problem is being solved), to modify the original problem at each iteration so that it behaves more like this model problem, and hence is easier to solve. Because preconditioning is an idea of such general usefulness, it will be discussed in considerably more detail in Chapter 5.

Efforts have also been made to extend conjugate-gradient methods towards Newton's method. This work comes under the category of truncated-Newton methods, and is the main subject of this thesis. The basic theory behind these algorithms will be outlined in the next two chapters.

3 Truncated-Newton Methods

3.1. Introduction

In the last chapter, we discussed the three traditional classes of methods for solving the basic unconstrained minimization problem (1.2.1). All the methods, as we saw, were based on the solution of the Newton equations (2.2.3). Truncated-Newton methods are no exception. In a sense, they are the converse to quasi-Newton methods. Quasi-Newton methods compute a search direction by exactly solving an approximation to the Newton equations, whereas truncated-Newton methods do so by approximately solving the exact Newton equations.

There are certain advantages to this approach. First of all, we are always dealing with exact second-derivative information; in other words, the sub-problem we are solving is more closely related to the actual problem we are interested in. Secondly, since we only need an approximate solution to (2.2.3), we can use an iterative method to solve it. Iterative methods generally have very low storage requirements, and do not explicitly require the Hessian matrix.

In the next section, we will briefly describe these methods, mention some of the iterative methods that have been proposed for solving (2.2.3), and indicate some of the problems that can arise. In the final sections, the linear conjugate-gradient and related algorithms will be derived via the Lanczos algorithm, and it will be shown how to use these algorithms to overcome the above-mentioned problems. Further aspects of this class of algorithms will be left to later chapters.

3.2. Basic Description of the Method

Because Newton's method is based on a Taylor series expansion near the solution of the minimization problem (1.2.1), there is no guarantee that the search direction it computes will be as crucial far away from x^*. In fact, at the beginning of the solution process, a reasonable approximation to the Newton direction may be almost as effective as the Newton direction itself. It is only gradually, as the solution is approached, that the Newton direction takes on more and more meaning.

This suggests using an iterative method to solve the Newton equations. Moreover, it should be an iterative method with a variable tolerance, so that far away from the

solution, (2.2.3) is not solved to undue accuracy. Only when the solution is approached should we consider expending enough effort to compute something like the exact Newton direction. As we approach the solution, the systems of equations we are required to solve become progressively more similar. Consequently it is possible that a closer approximation to the exact solution can be determined with no increase in effort by utilizing past information.

Sherman [1978] suggested using Successive-Over-Relaxation (SOR). This is the simplest of a whole class of methods that have been found to be effective for solving linear systems arising in partial differential equations. However, it is difficult to get SOR methods to perform well on general problems. Also, they appear to be prohibitively expensive to use in the context of truncated-Newton methods. The number of linear sub-iterations required to achieve superlinear convergence increases exponentially at each non-linear iteration.

Much better suited for this application is the linear conjugate-gradient algorithm. Although it is ideal for problems where the coefficient matrix has only a few distinct eigenvalues, it is guaranteed to converge (in exact arithmetic) in at most n iterations for any matrix. Thus, the type of exponential growth mentioned above for SOR-type methods is impossible, at least theoretically. Also, it can be shown that the linear conjugate-gradient algorithm is optimal in a sense to be defined later.

A requirement of both of these methods is that the coefficient matrix must be positive-definite. As remarked earlier, the Hessian matrix is only guaranteed to be positive semi-definite at the solution and may be indefinite elsewhere. Thus, whatever iterative method chosen to solve (2.2.3), it must be able to detect and cope with indefinite systems. This is very closely related to the situation with Newton's method, but in this case, we are not planning to perform a Cholesky factorization of the Hessian matrix, making it difficult to modify it directly. The next two sections will describe how to circumvent this problem in the case of the linear conjugate-gradient algorithm. Because the SOR-based methods are prohibitively expensive to use, even in ideal circumstances, they will not be considered further.

Paige and Saunders [1975] have developed two conjugate-gradient-like algorithms for dealing with symmetric indefinite systems of equations. The first of these, SYMMLQ, is identical to the traditional conjugate-gradient method in the positive-definite case, and

is not of much interest in this context. The second, MINRES, is based on minimizing the norm of the residual at each iteration. It produces different iterates than the CG method, and has many properties of value to us here. It will be discussed in section 3.7.

Finally, we give here a description of a truncated-Newton method in algorithmic form. The details of the methods used to iteratively solve the Newton equations and to precondition the algorithm will be given later.

(3.2.1) Truncated-Newton Algorithm

TN1. Given $x^{(k)}$, some approximation to x^*.

TN2. If $x^{(k)}$ is a sufficiently accurate approximation to the minimum of F, terminate the algorithm.

TN3. Approximately solve the Newton equations (2.2.3) using some iterative algorithm with preconditioning $M^{(k)}$. $M^{(k)}$ is chosen with information from some non-linear algorithm or from previous iterations (see Chapter 5).

TN4. Using the search direction computed in step **TN3**, use the line search algorithm (1.4.3) to compute a new point $x^{(k+1)}$. Go to step **TN2**.

3.3 The Linear Conjugate-Gradient Algorithm

A well known technique for the solution of large systems of linear equations is the linear conjugate-gradient method of Hestenes and Stiefel [1952]. This method can be directly applied to the Newton equations (2.2.3). The linear conjugate-gradient algorithm is particularly appropriate when matrix-vector products of the form $G^{(k)}v$ can be computed even though the matrix $G^{(k)}$ or its factorization cannot. The conjugate-gradient algorithm is usually derived as a direct method, in the sense that, theoretically, the solution is found after n iterations or less. However, in practice the algorithm behaves more like an iterative method since it computes a sequence of improving estimates and has the potential of converging in much more than n iterations. The finite termination properties of the conjugate-gradient algorithm are based on orthogonality relations which are not valid in finite precision arithmetic. It is possible to perform extra computations in order to recover finite termination, but this is expensive both in terms of storage and in terms of operation counts. For large problems, this is impractical. Recent work of

Parlett [1980] and others has attempted to overcome this difficulty through the use of selective reorthogonalization.

Although most of the literature on this algorithm refers to the solution of the equation $Ax = b$, it is felt that using this notation here would lead to unnecessary confusion. In order to be consistent with the other parts of this thesis, we shall solve the system of linear equations

$$Gp = b, \tag{3.3.1}$$

where G is an $n \times n$ positive-definite matrix. We shall use $\{p_q\}$ to denote members of an iterative sequence intended to solve (3.3.1). It will be assumed that all the operations are performed in exact arithmetic.

The conjugate-gradient algorithm can be derived by finding iterates that minimize the quadratic function $Q(p) = \frac{1}{2}p^TGp - b^Tp$.

Let p_q be the q-th approximation to the minimum of $Q(x)$ and let $v_1, v_2, \ldots, v_q$ be q linearly independent vectors that span a subspace $\mathcal{V}_q$. The minimum of $Q(x)$ may be computed by minimizing $Q(x)$ over an expanding sequence of linear manifolds that eventually contains $\Re^n$. If V_q denotes the matrix with columns $v_1, v_2, \ldots, v_q$ then the minimum of $Q(x)$ over the manifold $p_q + \mathcal{V}_q$ is given by the solution of the problem

$$\min_{w \in \Re^q} Q(p_q + V_q w).$$

If $p_q + V_q w$ is substituted into the quadratic function we find that the optimal w minimizes the function

$$\tfrac{1}{2}w^TV_q^TGV_qw + w^TV_q^Tr_q,$$

where $r_q = \nabla Q(p_q) = Gp_q - b$. This quadratic function has a minimum at the point $-(V_q^TGV_q)^{-1}V_q^Tr_q$ and consequently, the required minimum over the subspace is given by

$$p_{q+1} = p_q - V_q(V_q^TGV_q)^{-1}V_q^Tr_q.$$

Note that r_{q+1}, the gradient of $Q(p)$ at p_{q+1}, is orthogonal to the columns of V_q since

$$\begin{aligned} V_q^Tr_{q+1} &= V_q^T(Gp_{q+1} - b) \\ &= -V_q^TGV_q(V_q^TGV_q)^{-1}V_q^Tr_q + V_q^Tr_q \\ &= 0. \end{aligned}$$

The definition of p_{q+1} as a minimum over the manifold $p_q + \mathcal{V}_q$ has special significance

if each previous iterate p_j is obtained as the minimum over $p_{j-1} + \mathcal{V}_{j-1}$. In this case r_q will be orthogonal to all the columns of V_q except the last, and the minimum on the subspace $\mathcal{V}_q$ is given by

$$\begin{aligned} p_{q+1} &= p_q - V_q(V_q^T G V_q)^{-1} V_q^T r_q \\ &= p_q + \gamma V_q (V_q^T G V_q)^{-1} e_q, \end{aligned} \tag{3.3.2}$$

where $\gamma = -r_q^T v_q$ and e_q is the q-th column of the identity matrix.

Suppose that the columns of V_q are defined by the Lanczos recurrence relations (Lanczos [1950]). In this case we start with some vector v_1, $v_1^T v_1 = 1$, and form

$$\beta_{j+1} v_{j+1} = G v_j - \alpha_j v_j - \beta_j v_{j-1}, \quad \alpha_j = v_j^T G v_j, \tag{3.3.3}$$

where $v_0 = 0$, and β_{j+1} ($\beta_{j+1} \geq 0$) is chosen so that $\|v_{j+1}\|_2 = 1$. After the q-th step

$$G V_q = V_q T_q + \beta_{q+1} v_{q+1} e_q^T, \tag{3.3.4}$$

where

$$T_q = \begin{pmatrix} \alpha_1 & \beta_2 & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \beta_q \\ & & & & \beta_q & \alpha_q \end{pmatrix},$$

$$V_q^T V_q = I_q, \quad \text{and} \quad V_q^T v_{q+1} = 0.$$

The process will be terminated at the first zero β_j, so that in general we assume that β_j is nonzero for $j = 1, 2, \ldots, q$. In this case $V_q^T G V_q = T_q$ and (3.3.2) becomes

$$\begin{aligned} p_{q+1} &= p_q + \gamma V_q T_q^{-1} e_q \\ &= p_q + \gamma V_q (L_q^T)^{-1} D_q^{-1} L_q^{-1} e_q, \end{aligned}$$

where L_q and D_q are the Cholesky factors of T_q. Since L_q has unit diagonal elements, $L_q^{-1} e_q = e_q$. Consequently,

$$\begin{aligned} p_{q+1} &= p_q + \frac{\gamma}{d_q} V_q (L_q^T)^{-1} e_q, \\ &= p_q + \sigma_q u_q, \end{aligned} \tag{3.3.5}$$

where $\sigma_q = -r_q^T v_q / d_q$ and u_q is given by the q-th column of the matrix

$$U_q = V_q (L_q^T)^{-1}.$$

Paige and Saunders [1975] show that the columns of U_q can be computed from the recurrence relations

$$u_1 = v_1, \qquad u_q = v_q - l_q u_{q-1},$$

where l_q is the $(q-1)$-th sub-diagonal element of the lower bi-diagonal matrix L_q.

When the vector v_1 used to start the Lanczos process is chosen as a multiple of the right-hand-side vector b and when x_1 is zero, this algorithm is mathematically equivalent to the Hestenes-Stiefel conjugate-gradient algorithm. The derivation here emphasizes the fact that a whole class of conjugate-direction methods can be generated from different choices for v_1.

It is well known that in general, rounding error seriously impairs the performance of Lanczos tri-diagonalization by causing a loss of orthogonality in the vectors $\{v_j\}$. This implies that the matrices $V_q^T G V_q$ will no longer be tri-diagonal and the solution p_{q+1} over the subspace $\mathcal{V}_q$ will be correspondingly inaccurate. The effects of this error are noticably reduced if the starting vector v_1 is taken to be a multiple of b. The reason for this is that, in the Paige-Saunders algorithm, each p_q is algebraically of the form $V_q y$, where $T_q y = \beta_1 e_1$. From (3.3.4) we have

$$G V_q y = V_q T_q y + \beta_{q+1} v_{q+1} e_q^T y$$

to working precision, and consequently,

$$\begin{aligned} G p_q &= \beta_1 V_q e_1 + \beta_{q+1} v_{q+1} e_q^T y \\ &= b + \beta_{q+1} v_{q+1} e_q^T y. \end{aligned} \tag{3.3.6}$$

This expression does not depend at all upon the orthogonality of the Lanczos vectors and indicates that p_q will be the solution of a problem with right-hand side that differs from the true b by $\beta_{q+1} v_{q+1} e_q^T y$, a quantity that will ultimately be sufficiently small. Unfortunately, a relationship analogous to (3.3.6) does not hold for arbitrary v_1.

In the light of these remarks, we shall always use the term *Lanczos iteration* to refer to the recurrence relations (3.3.3) with v_1 defined as a multiple of b. Although this is mathematically equivalent to the linear conjugate-gradient algorithm of Hestenes and

Steifel, this derivation allows us to modify the algorithm in the case when the Hessian matrix G is not positive definite.

3.4. Indefinite Systems

When the matrix G is indefinite, Paige and Saunders note that the conjugate-gradient method is unstable and propose a modified Lanczos method based on the LQ factorization of T_q rather than the Cholesky factorization. In the context of minimization, however, even the exact solution of an indefinite system is of little practical value since the resulting direction of search is likely to be a non-descent direction. As in the case of modified-Newton methods for the factorization of $G^{(k)}$, we can make better use of the solution of a neighboring positive-definite system.

The proposed method is based on the following theorem. We shall assume that the Lanczos iteration, when applied with exact arithmetic to a symmetric matrix G, terminates at iteration s ($s \leq n$), i.e., β_{s+1} vanishes. As in the last section, we shall use T_q and V_q to denote the $q \times q$ and $n \times q$ matrices associated with the q-th stage of the Lanczos iteration.

Theorem (3.4.1) Let $E_s = \text{diag}(e_{11}, e_{22}, \ldots, e_{ss})$ be a diagonal matrix with non-negative entries and let E_q denote the $q \times q$ principal sub-matrix of E_s. If the matrix $T_q + E_q$ is positive definite with Cholesky factors L_q and D_q, the iteration

$$\hat{p}_1 = 0, \qquad \hat{p}_{q+1} = \hat{p}_q + \sigma_q V_q L_q^{-T} e_q,$$

where $\sigma_q = -v_q^T r_q / d_q$, solves the linear system,

$$(G + V_s E_s V_s^T)\hat{p} = b.$$

Proof Let $\hat{G}$ denote the matrix $G + V_s E_s V_s^T$. Using the orthogonality of the Lanczos vectors

$$\begin{aligned} V_q^T \hat{G} V_q &= V_q^T G V_q + E_q \\ &= T_q + E_q \\ &= L_q D_q L_q^T. \end{aligned} \tag{3.4.2}$$

We can apply the ideas of Section 3.3 to generate the sequence $\{\hat{p}_j\}$ defined by the

recurrence relations

$$\begin{aligned} \hat{r}_q &= \hat{G}\hat{p}_q - b \\ \hat{p}_{q+1} &= \hat{p}_q - V_q(V_q^T\hat{G}V_q)^{-1}V_q^T\hat{r}_q, \end{aligned} \tag{3.4.3}$$

that will solve the linear system $\hat{G}\hat{p} = b$.

Substituting the expression (3.4.2) for $V_q^T\hat{G}V_q$ in (3.3.5) and using a similar analysis to that used to obtain (7) we find

$$\hat{p}_{q+1} = \hat{p}_q + \sigma_q V_q L_q^{-T} e_q,$$

where $\sigma_q = -v_q^T\hat{r}_q/d_q$. The scalar $v_q^T\hat{r}_q$ is computed from r_q using

$$\begin{aligned} v_q^T\hat{r}_q &= v_q^T(\hat{G}\hat{p}_q - b) \\ &= v_q^T(G\hat{p}_q - b) + v_q^T V_q E_q V_q^T\hat{p}_q \\ &= v_q^T r_q + e_{qq}v_q^T\hat{p}_q. \end{aligned}$$

Since $\hat{p}$ is in the span of $\{v_1, \ldots, v_{q-1}\}$, and V_q is an orthogonal matrix, the second term on the right-hand side vanishes. This proves the theorem. ∎

Corollary (3.4.4) The vector $\hat{p}$ obtained from the recurrence relations defined in Theorem 1 satisfies the positive-definite system

$$(G + VEV^T)\hat{p} = b,$$

where $\|VEV^T\| = \|E\|$.

Proof If s is equal to n the corollary follows trivially. If $s < n$ then V will be the matrix $(V_s\ \bar{V})$, where $\bar{V}$ is the orthogonal complement of V_s. The remaining $n - s$ diagonal elements of E are arbitrary and may be chosen so that $A + VEV^T$ is positive definite. ∎

The modified Lanczos method may be applied directly to the Newton equations (2.2.3). When exact arithmetic is used and the Lanczos iteration is continued until β_{q+1} is zero (i.e., q steps are computed), the nonlinear algorithm is a modified Newton method with a positive-definite approximate Hessian

$$G^{(k)} + \Omega^{(k)},$$

where $\Omega^{(k)}$ is the matrix $V_q E_q V_q^T$. Note that, unlike the modification produced by the

direct application of the modified Cholesky factorization, $\Omega^{(k)}$ is not a diagonal matrix. The orthogonality of the Lanczos vectors implies that

$$\|V_q E_q V_q^T\| = \|E_q\|.$$

Consequently, when the diagonal modification to T_q is small, the modification to $G^{(k)}$ will be small also.

The *truncated Newton method* of Dembo and Steihaug [1980] "solves" (2.2.3) by performing a *limited number* of iterations of the linear conjugate-gradient method. The iterations are terminated ("truncated") before the system is solved exactly. The final iterate of the truncated sequence is then taken as an approximate solution of (2.2.3). If a single linear iteration is used, $p^{(k)}$ will be the steepest-descent direction $-g^{(k)}$; if the sequence is not truncated, $p^{(k)}$ will be the solution of (2.2.3). Thus, the algorithm computes a vector that interpolates between the steepest-descent direction and the Newton direction.

Dembo and Steihaug showed that, if $G^{(k)}$ is positive definite and the initial iterate of the linear conjugate-gradient scheme is the steepest-descent direction $-g^{(k)}$, all succeeding linear iterates will be directions of descent with respect to $F(x)$.

When $G^{(k)}$ is not guaranteed to be positive definite, $p^{(k)}$ may not be a descent direction. We propose that the modified Lanczos algorithm be used to compute the direction of search. The following theorem indicates that the direction of search obtained by terminating the modified Lanczos scheme will always be a direction of descent, irrespective of the definiteness of $G^{(k)}$.

Theorem (3.4.5) Let $\{p_q\}$ denote the sequence of iterates computed by the modified Lanczos algorithm with p_0 equal to the zero vector, and assume that $g^{(k)} \neq 0$. Then ${g^{(k)}}^T p_q < 0$ for all $q > 0$.

Proof When p_0 is zero, p_q $(q > 0)$ is the solution of the minimization problem

$$\min_{p \in \mathcal{V}_{q-1}} \tfrac{1}{2} p^T (G^{(k)} + \Omega^{(k)}) p + {g^{(k)}}^T p$$

where $\Omega^{(k)}$ is some modification to $G^{(k)}$ chosen so that $G^{(k)} + \Omega^{(k)}$ is positive definite. Thus

$$p_q = -V_{q-1} \left(V_{q-1}^T (G^{(k)} + \Omega^{(k)}) V_{q-1} \right)^{-1} V_{q-1} g^{(k)}.$$

Direct pre-multiplication by $g^{(k)^T}$ gives

$$g^{(k)^T} p_q = -g^{(k)^T} V_{q-1} \left(V_{q-1}^T (G^{(k)} + \Omega^{(k)}) V_{q-1} \right)^{-1} V_{q-1}^T g^{(k)} < 0,$$

since by construction, $V_{q-1}^T (G^{(k)} + \Omega^{(k)}) V_{q-1}$ is positive definite. ∎

3.5. Computing a Modified Factorizaton of a Tridiagonal Matrix

At the first stage of the Lanczos process we need to find the Cholesky factorization of the 1×1 "matrix" given by

$$T_1 = (\alpha_1).$$

If A is not positive definite, α_1 may be negative or zero, and it is replaced by

$$\bar{\alpha}_1 = \max \{\alpha_1, \delta\},$$

where δ is a pre-assigned small positive constant. This is an allowable "diagonal" modification to T_1 since

$$\bar{\alpha}_1 = \alpha_1 + \rho_1,$$

where $\rho_1 = \max \{0, \delta - \alpha_1\}$. The introduction of the constant δ is necessary to bound the factorization away from singularity. Usually δ will be a multiple of the relative machine precision.

At the second stage of the algorithm we need the modified factorization of the matrix

$$\begin{pmatrix} \bar{\alpha}_1 & \beta_2 \\ \beta_2 & \alpha_2 \end{pmatrix}. \tag{3.5.1}$$

Let the Cholesky factors of this matrix be written as

$$\begin{pmatrix} 1 & 0 \\ l_2 & 1 \end{pmatrix} \begin{pmatrix} d_1 & 0 \\ 0 & d_2 \end{pmatrix} \begin{pmatrix} 1 & l_2 \\ 0 & 1 \end{pmatrix}.$$

Straightforward application of the Cholesky factorization to (3.5.1) gives $d_1 = \bar{\alpha}_1$, $d_2 = \max\{\delta, \phi\}$, and $l_2 = \beta_2 / \bar{\alpha}_1$, where $\phi = \alpha_2 - \beta_2^2 / \bar{\alpha}_1$. If ϕ is negative, the matrix is indefinite and a diagonal correction must be made in order to ensure sufficient positive definiteness. The smallest modification to the $(2,2)$ element of (3.5.1) that maintains positive definiteness results from redefining d_2 as δ. This modification adds the quantity $-\phi + \delta$ to the second diagonal element of (3.5.1). Unfortunately, this algorithm has a

serious disadvantage, as the following example will illustrate.

Consider the Cholesky factorization of the matrix

$$T = \begin{pmatrix} \delta & 1 \\ 1 & 1 \end{pmatrix}. \tag{3.5.2}$$

In this case, ϕ is the large negative quantity $1 - 1/\delta$ and the diagonal d_2 is set to δ. This gives the factorization

$$\begin{pmatrix} 1 & 0 \\ \frac{1}{\delta} & 1 \end{pmatrix}\begin{pmatrix} \delta & 0 \\ 0 & \delta \end{pmatrix}\begin{pmatrix} 1 & \frac{1}{\delta} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \delta & 1 \\ 1 & \delta + \frac{1}{\delta} \end{pmatrix}.$$

Note that, although we have made the smallest allowable modification to d_2, we have produced a very large diagonal correction and the modified matrix is in no way "close" to the original. This has occurred because the lower-triangular element has been allowed to become large.

This problem does not arise when using the Gill-Murray modified Cholesky factorization (Gill and Murray [1974a]) since the diagonals are adjusted so as to give lower-triangular elements that are bounded in magnitude by an *a priori* bound. However, in order to compute this bound it is necessary to determine an accurate bound on $\|G\|$ which may not be possible or convenient if G is large and not stored explicitly.

An alternative to the Gill-Murray factorization which is, nevertheless, similar in principle requires the computation of the factorization

$$\begin{pmatrix} \bar{\alpha}_1 & \beta_2 \\ \beta_2 & \alpha_2 \end{pmatrix} + \begin{pmatrix} \sigma_1 & 0 \\ 0 & \rho_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_2 & 1 \end{pmatrix}\begin{pmatrix} \bar{d}_1 & 0 \\ 0 & d_2 \end{pmatrix}\begin{pmatrix} 1 & l_2 \\ 0 & 1 \end{pmatrix},$$

so that the quantity $\sigma_1 + \rho_2$ is minimized. (We use a different notation for the elements of the diagonal modification and the diagonal factor because ρ_2 and d_2 may be modified again at the next stage of the factorization.) Thus if we define the quantity

$$\tau = \max\{\phi, \delta\},$$

we need to solve the problem

$$\begin{aligned} \min \quad & \sigma_1 + \rho_2 \\ \text{subject to} \quad & \tau = \alpha_2 + \rho_2 - \beta_2^2/(\bar{\alpha}_1 + \sigma_1) \\ & \sigma_1 \geq 0, \quad \rho_2 \geq 0. \end{aligned}$$

If $\phi \geq \delta$, $\sigma_1 = \rho_2 = 0$. Otherwise $\tau = \delta$ and we compute,

$$\sigma_1 = |\beta_2| - \bar{\alpha}_1, \quad \text{and} \quad \rho_2 = \delta - \alpha_2 + |\beta_2|, \tag{3.5.3}$$

which are the optimal values of σ_1 and ρ_2 ignoring the non-negativity constraints. If either correction is negative we use

$$\sigma_1 = 0 \quad \text{and} \quad \rho_2 = \delta - \alpha_2 + \beta_2^2/\bar{\alpha}_1 = \delta - \phi, \tag{3.5.4}$$

or

$$\sigma_1 = \beta_2^2/(\alpha_2 - \delta) - \bar{\alpha}_1, \quad \text{and} \quad \rho_2 = 0, \tag{3.5.5}$$

choosing the sensible pair that minimizes $\sigma_1 + \rho_2$. Since $\phi < \delta$, at least (3.5.4) must be feasible.

When this modified Cholesky factorization is used on the matrix (3.5.2), the factors are given by

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & \delta \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1+\delta \end{pmatrix},$$

which is suitably "close" to the original matrix.

The first column of the Cholesky factorization is unaffected by subsequent iterations and consequently l_2 is the required sub-diagonal element of L with

$$e_{11} = \rho_1 + \sigma_1.$$

The next stage of the reduction involves the matrix

$$\begin{pmatrix} \bar{\alpha}_2 & \beta_3 \\ \beta_3 & \alpha_3 \end{pmatrix},$$

where $\bar{\alpha}_2 = \alpha_2 + \rho_2$ and by construction, $\bar{\alpha}_2 \geq \delta$. This matrix is identical in structure to (3.5.1) and so the process can be continued to find the modified factorization of T_2, T_3, ... etc.

Two advantages of the Gill-Murray modified Cholesky factorization are that (a) it is possible to bound the diagonal modification, and (b) it is possible to compute a direction of negative curvature when $g^{(k)} = 0$ and $G^{(k)}$ is indefinite (see section 2.2). The implementation of the algorithm as described in their work requires accurate *a priori* bounds on the elements of the matrix $G^{(k)}$. In our case, the tridiagonal matrix T_q is being

factored as it is generated, and the matrix $G^{(k)}$ may not be available. It is still possible, though, to bound the norm of the modification and to compute directions of negative curvature when the gradient is zero.

Theorem (3.5.6) Let T be a symmetric tridiagonal matrix, with principal $i \times i$ submatrix T_i. Assume that a modified Cholesky factorization

$$T_i + E_i = L_i D_i L_i^T$$

is computed using the algorithm described above. If

$$\gamma_i = \max_{j \le i}\{|\alpha_j|\}$$
$$\varsigma_i = \max_{j \le i}\{|\beta_j|\},$$

where $\{\alpha_j\}$ and $\{\beta_j\}$ are the diagonal and subdiagonal elements of T, respectively, and if δ is a positive tolerance for zero, then

$$\|E_i\|_2 \le i[\delta + \gamma_i + \varsigma_i].$$

Proof (by induction) Initially, $T_1 = (\alpha_1)$. If $\alpha_1 > \delta$, then $E_1 = (0)$; otherwise, $E_1 = (\delta - \alpha_1)$. In either case, $\|E_1\| \le \delta + |\alpha_1| = 1 \cdot [\delta + \gamma_1 + \varsigma_1]$. (The norm used is the 2-norm.)

Assume that $\|E_i\| \le i[\delta + \gamma_i + \varsigma_i]$. If $\sigma_i = \rho_{i+1} = 0$, then no modification is necessary, $\|E_{i+1}\| = \|E_i\|$, and hence $\|E_{i+1}\| \le (i+1)[\delta + \gamma_{i+1} + \varsigma_{i+1}]$.

Otherwise, $\tau = \delta$ and we compute σ_i and ρ_{i+1} from (3.5.3), (3.5.4), or (3.5.5). When (3.5.3) is used,

$$\|E_{i+1}\| \le \|E_i\| + \delta + \gamma_{i+1} + \varsigma_{i+1},$$

and the result follows. If (3.5.3) is infeasible, we use either

$$\sigma_i + \rho_{i+1} = \beta_{i+1}^2/\bar{\alpha}_i - \tilde{\alpha}_{i+1} \equiv \psi$$

or

$$\sigma_i + \rho_{i+1} = \beta_{i+1}^2/\tilde{\alpha}_{i+1} - \bar{\alpha}_i \equiv \theta,$$

where $\tilde{\alpha}_{i+1} \equiv \alpha_{i+1} - \delta$. There are three cases:

(i) $\max\{\bar{\alpha}_i, \tilde{\alpha}_{i+1}\} \le |\beta_{i+1}|$: In this case, $|\beta_{i+1}| - \bar{\alpha}_i$ and $|\beta_{i+1}| - \tilde{\alpha}_{i+1}$ are both positive, so that (3.5.3) would have been feasible.

(ii) $|\beta_{i+1}| \leq \bar{\alpha}_i$: First, $\psi \leq |\beta_{i+1}| - \tilde{\alpha}_{i+1}$. Also, because (3.5.4) is feasible, we can conclude that $\psi \geq 0$.

(iii) $|\beta_{i+1}| \leq \tilde{\alpha}_{i+1}$: First, $\theta \leq |\beta_{i+1}| - \bar{\alpha}_i$. Because $\bar{\alpha}_i > 0$ (by construction) and $\tilde{\alpha}_{i+1} > 0$ (by assumption), then $\theta = (\bar{\alpha}_i/\tilde{\alpha}_{i+1})\psi \geq 0$.

We are trying to minimize $\sigma_i + \rho_{i+1} = \min\{\theta, \psi\}$, subject to the feasibility constraints. From the case-by-case analysis above, we can conclude that $\min\{\theta, \psi\} \leq \max\{|\beta_{i+1}| - \bar{\alpha}_i, |\beta_{i+1}| - \tilde{\alpha}_{i+1}\}$, and hence

$$\|E_{i+1}\| \leq \|E_i\| + \delta + \gamma_{i+1} + \zeta_{i+1}.$$

This implies that $\|E_{i+1}\| \leq (i+1)[\delta + \gamma_{i+1} + \zeta_{i+1}]$, and the result is proved.∎

In the above discussion, it was assumed that the modifications to the diagonal elements of the 2×2 matrices were both positive. When factoring a tridiagonal matrix, a diagonal element might be modified twice, and in this case it might be possible to allow one of these modifications to be negative. This would not affect the *a priori* bound on E derived above. For this reason, we have not examined this possibility further.

3.6. Computing a Direction of Negative Curvature

The procedure for the computation of the direction of search will break down if $\|g\|$ is zero. If G is positive definite, then a solution has been determined. It remains, however, to confirm that G is indeed positive definite. Moreover, if G is indefinite, further progress can be made by moving along a direction of negative curvature.

Suppose that $\|g\|$ is small. We wish to determine if G is indefinite, and if so, compute p such that $p^TGp < 0$. To do this, we choose v_1 randomly, $\|v_1\| = 1$, as the initial vector for the Lanczos process. The Lanczos iteration

$$V_j^TGV_j = T_j, \qquad T_j = L_jD_jL_j^T$$

is performed as long as T_j is positive semi-definite. (Parlett [1980] reports that good approximations to the extreme eigenvalues can be obtained in $2n^{\frac{1}{2}}$ iterations; if G is indefinite, one of the extreme eigenvalues will be negative.) Note that, because we are trying to determine if G is indefinite, the tolerance δ should be set to zero.

Assume that T_q, $q \leq 2n^{\frac{1}{2}}$, is the last positive semi-definite matrix that occurs. Also,

suppose for the moment that β_{q+1} is non-zero. Then $T_{q+1} = L_{q+1}\overline{D}_{q+1}L_{q+1}^T$, where

$$L_{q+1} = \begin{pmatrix} L_q & 0 \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad \overline{D}_{q+1} = \begin{pmatrix} D_q & b_{q+1} \\ b_{q+1}^T & \alpha_{q+1} \end{pmatrix},$$

where $b_{q+1}^T = (0, \ldots, 0, \beta_{q+1})$. Note that only the last diagonal element of D_q can be zero. In order to determine a direction of negative curvature, we perform an orthogonal spectral decomposition of $\begin{pmatrix} d_q & \beta_{q+1} \\ \beta_{q+1} & \alpha_{q+1} \end{pmatrix}$:

$$\begin{pmatrix} d_q & \beta_{q+1} \\ \beta_{q+1} & \alpha_{q+1} \end{pmatrix} = Q\begin{pmatrix} \lambda_q & 0 \\ 0 & -\lambda_{q+1} \end{pmatrix}Q^T, \qquad Q = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix}, \qquad Q^TQ = I,$$

with $\lambda_{q+1} > 0$. This is possible because T_{q+1} has exactly one negative eigenvalue. Let $p = u_{q+1}$, where

$$U_{q+1} = (u_1 \mid \cdots \mid u_{q+1}) = V_{q+1}\tilde{L}_{q+1}^{-T}, \qquad \tilde{L}_{q+1} = L_{q+1}\begin{pmatrix} I & 0 \\ 0 & Q \end{pmatrix};$$

$u_{q+1} = q_{12}u_q + q_{22}v_{q+1}$. This is nearly the same as the formula used to compute the linear conjugate-gradient search direction, so that no additional vector storage is required to implement this idea. Then, if $e_{q+1}^T = (0, \ldots, 0, 1)$ and $\tilde{D}_{q+1} = \text{diag}\{d_1, \ldots, d_{q-1}, \lambda_q, -\lambda_{q+1}\}$,

$$\begin{aligned} p^TGp &= e_{q+1}^TU_{q+1}^TGU_{q+1}e_{q+1} \\ &= e_{q+1}^T\tilde{L}_{q+1}^{-1}V_{q+1}^TGV_{q+1}\tilde{L}_{q+1}^{-T}e_{q+1} \\ &= e_{q+1}^T\tilde{L}_{q+1}^{-1}[\tilde{L}_{q+1}\tilde{D}_{q+1}\tilde{L}_{q+1}^T]\tilde{L}_{q+1}^{-T}e_{q+1} \\ &= -\lambda_{q+1} < 0. \end{aligned}$$

Thus p is a direction of negative curvature. It should be noted that, since the Lanczos algorithm seeks out the most negative eigenvalue of G, the direction p computed in this way ought to be an excellent direction of negative curvature, i.e. $p^TGp/\|p\|$ will be close to its minimum value.

Suppose now that $\beta_{q+1} = 0$. If T_q is positive definite, and provided our initial random vector does not lie wholly in the space spanned by the eigenvectors corresponding to positive eigenvalues, then G is positive definite and we are at a local minimum of the objective function F. However, if the initial vector does lie in the positive eigenspace, no such conclusion can be made. To guarantee the indefiniteness of G, we require a more complex procedure.

We will carry out the Lanczos procedure using a series of initial vectors. The first,

v_1^1, will be chosen randomly as above. If the Lanczos algorithm terminates at iteration q, with $q < n$, and T_q is positive definite, then we choose another initial vector v_1^2 orthogonal to $\{v_1^1, Gv_1^1, \ldots\}$ and run the Lanczos algorithm again. We continue in this way until either we encounter an indefinite T_q, or until $\{v_1^1, Gv_1^1, \ldots, v_1^2, Gv_1^2, \ldots\}$ spans $\Re^n$. In the latter case, we can assert that G is positive definite.

This procedure is impractical for large problems. In the worst case, it will take a full n steps to determine whether G is indefinite. Suppose that G has one negative eigenvalue. Also, let $\{v_1^i\}$ be an orthogonal set of eigenvectors for G, with v_1^n corresponding to the negative eigenvalue. Then the Lanczos algorithm will converge in one iteration for each v_1^i, and all n starting vectors will have to be used. When n is large, this is unsuitably expensive.

If G is indefinite then, due to the influence of rounding errors, it is highly unlikely that the Lanczos algorithm will terminate without discovering the indefiniteness in G. Even if the initial vector v_1 contains no component in the negative eigenspace, any rounding error would almost certainly introduce one. This would then allow a negative eigenvalue to develop in T_q as desired. Thus, the worst case behavior described above is unlikely to occur in practice. This justifies using a single starting vector when seeking a direction of negative curvature.

If T_q is only positive semi-definite, i.e. d_q=0, then it is not possible to determine a direction of negative curvature for the quadratic approximation. If we set $p = u_q$, the q-th linear conjugate-gradient search direction, then $p^T G p = 0$ using an argument similar to that above. Such a p may be a direction of negative curvature for F, even though it is not for the quadratic subproblem. This would depend on the higher-order derivatives of F.

3.7. Minimum Residual Methods

As was seen above, the preconditioned Lanczos method generates a tridiagonal matrix as a projection of the coefficient matrix in (3.3.1). In the previous sections, we used this tridiagonal matrix to minimize a quadratic function related to our original system of equations. This tridiagonal matrix can also be used to implicitly solve the normal equations

$$G^T G p = G^T b.$$

This idea is the basis of the minimum residual algorithm MINRES of Paige and Saunders [1975].

When minimizing the quadratic function, we formed an LDL^T factorization of the tridiagonal matrix T. In this context, it is more appropriate to factor T_q (or, to be more exact, $T_q + E_q$) as

$$T_q = \bar{L}_q Q_q, \qquad Q_q^T Q_q = I, \tag{3.7.1}$$

with $\bar{L}_q$ lower triangular. The bar is used to indicate that $\bar{L}_q$ differs from the $q \times q$ leading part of $\bar{L}_{q+1}$ in the (q,q) element only. The details of this factorization can be found in the Paige and Saunders paper.

If we carry out the orthogonal factorization, we obtain that

$$V_q^T G^2 V_q = T_q^2 + \beta_{q+1}^2 e_q e_q^T = \bar{L}_q \bar{L}_q^T + \beta_{q+1}^2 e_q e_q^T = L_q L_q^T, \tag{3.7.2}$$

where L_q is the leading q by q part of $\bar{L}_{q+1}$. If we project the normal equations onto the space spanned by the columns of the matrix V_q, we obtain

$$V_q^T G^2 V_q z_q = V_q^T G b, \qquad p_q = V_q z_q.$$

The right-hand side of this system can be written as

$$V_q^T G b = \beta_1 V_q^T G v_1 = \beta_1 T_q e_1.$$

Using (3.7.1) and (3.7.2), we obtain the following system of equations for z_q:

$$L_q L_q^T z_q = \beta_1 \bar{L}_q Q_q e_1. \tag{3.7.3}$$

But we can write

$$\bar{L}_q = L_q D_q, \qquad D_q \equiv \operatorname{diag}(1, 1, \ldots, c_q),$$

and while L_q is nonsingular, (3.7.3) gives

$$\begin{aligned} L_q^T z_q &= \beta_1 D_q Q_q e_1 \equiv (\tau_1, \ldots, \tau_q)^T \equiv t_q, \\ \tau_1 &\equiv \beta_1 c_1, \qquad \tau_i \equiv \beta_1 s_1 s_2 \cdots s_{i-1} c_i, \quad i = 2, \ldots, q, \end{aligned} \tag{3.7.4}$$

so there is minimal error in computing $L_q^T u_q$. Clearly z_q cannot be found until the algorithm is completed, but it is not really needed; instead we form

$$N_q \equiv [n_1, \ldots, n_q] = V_q L_q^{-T}$$

column by column, and then

$$x_q^R = V_q z_q = V_q L_q^{-T} L_q^T z_q = N_q t_q,$$

where t_q is developed in (3.7.2), and the superscript ^{R} shows that this is the vector which gives the minimum residual. It can be seen that this formula does not require the storage of the matrix N; all that is needed is its final column.

Because we will wish to terminate this algorithm based on the norm of the residual, it is important that this quantity can be computed with little effort and without requiring excess storage. It follows from the results of Paige and Saunders that

$$\|r_q^R\|_M = |\beta_1 s_1 s_2 \cdots s_q|,$$

which satisfies our requirements, and which shows clearly how the residual norm decreases each step. Here, $\|\cdot\|_M$ refers to the M-norm of a vector, where M is the preconditioning matrix for the Lanczos algorithm (see section 3.8 below).

3.8. Preconditioning the Lanczos Algorithm

When exact arithmetic is used throughout, the number of iterations required to solve a linear system $Gp = b$ using the conjugate-gradient method or the MINRES algorithm is equal to the number of distinct eigenvalues of G (see, for example, Luenberger [1973], pp. 176–178). Therefore, the performance should be significantly improved when the original system is replaced by an equivalent system in which the matrix has many unit eigenvalues. The purpose of *preconditioning* is to construct a transformation to have this effect.

Let M be a symmetric, positive-definite matrix. The solution of $Gp = b$ can be found by solving the system

$$M^{-\frac{1}{2}} G M^{-\frac{1}{2}} y = M^{-\frac{1}{2}} b,$$

and forming $p = M^{-\frac{1}{2}} y$. Let R denote the matrix $M^{-\frac{1}{2}} G M^{-\frac{1}{2}}$; we have $M^{-\frac{1}{2}} R M^{\frac{1}{2}} = M^{-1} G$ and therefore R is similar to $M^{-1}G$ and has the same eigenvalues. The idea is to choose M so that as many of the eigenvalues of $M^{-1}G$ as possible are close to unity. This is roughly equivalent to choosing M so that the condition number of $M^{-1}G$ is as small as possible; the matrix M is known as the *preconditioning matrix*.

Given a preconditioning matrix M, we can apply the Lanczos algorithm to the transformed system without forming R and without the need to find the square root of the matrix M. In practice, M will often not be explicitly available. It will only be available as an operator, and all that will be possible is to solve systems of linear equations with coefficient matrix M.

The recurrence relations analogous to (3.3.3) for the transformed system are

$$\beta_{j+1}v_{j+1} = M^{-1}Gv_j - \alpha_j v_j - \beta_j v_{j-1}, \qquad \alpha_j = v_j^T G v_j,$$

where v_0 and v_1 are chosen as before. Notice, however, that for the preconditioned algorithm the vectors v_i are normalized so that $\|v_i\|_M = 1$. After the q-th step we have

$$GV_q = MV_qT_q + \beta_{q+1}Mv_{q+1}e_q^T.$$

Note that the matrices M, T_q, and V_q satisfy the relations

$$V_q^T M V_q = I_q, \qquad \text{and} \qquad V_q^T G V_q = T_q.$$

This preconditioned Lanczos algorithm allows us to solve the system of equations (3.3.1) in the same way as in sections 3.3 or 3.7. The crucial fact in the derivation in those sections is that the matrix V_q transforms the matrix G to tridiagonal form. This is still true for the preconditioned algorithm.

We shall discuss the choice of preconditioning matrix in chapter 5. We shall be particularly interested in using a matrix M that is an approximation to the inverse of G. This matrix can be obtained using information from a nonlinear conjugate-gradient-type method together with information from previous linear subiterations.

4 Terminating the Linear Algorithm

4.1. Introduction

In order to fully define the simplest form of a truncated-Newton algorithm, all that remains is to state how to terminate the linear algorithm. The fundamental results in this area were proved by Dembo, Eisenstadt and Steihaug [1980]. They provide very useful guidelines for developing practical convergence criteria.

In section 4.2, the results of Dembo et al. will be described from the point of view of function minimization (they were originally stated in the context of solving systems of non-linear equations). Because these results are stated in terms of the 2-norm of the residual in (2.2.3), they are not directly applicable or even natural when used in conjunction with methods based on minimizing a quadratic function. More useful extensions of their results will be proved in section 4.3. In section 4.4, practical stopping criteria for the linear algorithm will be discussed. Finally, in section 4.5, truncated-Newton algorithms based on a trust-region approach will be derived.

4.2. Termination Based on $\|r^{(k)}\|_2$

Actually, the results in this section involve the relative residual

$$r^{(k)}/\|g^{(k)}\|, \tag{4.2.1}$$

where

$$r^{(k)} \equiv G^{(k)}p^{(k)} + g^{(k)}.$$

The relative residual in (4.2.1) is used because it is scale free, i.e. multiplying the objective function F by a constant does not affect its value. Note that since $\|g^{(k)}\| \to 0$ and $\|p^{(k)}\| \to 0$, then $\|p^{(k)} - (-g^{(k)})\| \to 0$. Since $-g^{(k)}$ would be an exceedingly poor approximation to $p^{(k)}$ to use within the algorithm, clearly it is necessary to scale in the manner described.

All the results of this section will be of the following form: the linear iteration will be truncated if the current estimate $p_q^{(k)}$ of the search direction guarantees that

$$\frac{\|r^{(k)}\|}{\|g^{(k)}\|} \leq \phi_k, \qquad k = 0, 1, \ldots, \tag{4.2.2}$$

where $\{\phi_k\}$ is some "forcing sequence". The algorithm is then completely defined given that the sequence $\{\phi_k\}$ has been specified.

Before proceeding with the main result, we require the following definition.

Definition (4.2.3) $G(x)$ is *Hölder continuous* at x^* if there exist constants $p \in (0, 1]$ and L so that for all y in a neighborhood of x^*

$$\|G(y) - G(x^*)\| \leq L\|y - x^*\|^p.$$

We are now in a position to state

Theorem (4.2.4) Assume that $F : \Re^n \to \Re$ is a real-valued function such that

1) There exists a local minimum x^* of F.

2) F is twice continuously differentiable in a neighborhood of x^*

3) $G(x^*)$ is nonsingular (and hence positive definite)

and that the truncated-Newton sequence $\{x^{(k)}\}$ converges to x^*. Then

i) if $\lim_{k\to\infty} \phi_k = 0$, the convergence rate of $\{r^{(k)}\}$ and $\{x^{(k)}\}$ will be superlinear.

In addition, if $G(x)$ is Hölder continuous at x^* with exponent $p \in (0, 1]$, then for some $c > 0$,

ii) if $\phi_k \leq c\|g^{(k)}\|^p$, the sequence $\{x^{(k)}\}$ converges with Q-order $(1 + p)$; and

iii) if $\{\phi_k\}$ converges to 0 with R-order $(1 + p)$, then $\{x^{(k)}\}$ converges to x^* with R-order $(1 + p)$.∎

The proofs of these results can be found in Dembo et al. [1980].

4.3. Alternative Assessment Criteria

In order to approximately solve the system of linear equations (2.2.3), some variant of the conjugate-gradient algorithm is used. Although Theorem (4.2.4) is useful in indicating when to stop the conjugate-gradient iteration, it is based on $\|r_q\|$ a quantity which does not decrease monotonically as the algorithm progresses (the subscript q refers to the linear conjugate-gradient iteration). The algorithm is based upon the minimization of the quadratic function

$$Q(p) = \tfrac{1}{2}p^T Gp + p^T g.$$

It would be preferable to stop the algorithm based on the value of this quadratic function, since it is a closer measure of how the conjugate-gradient algorithm is converging. Ideally, we would like to measure convergence in terms of the quantity

$$\frac{\|p-\overline{p}\|}{\|g\|},$$

where $\overline{p}$ is the minimizing point for the quadratic function $Q(p)$. Since this is unavailable during the computation, this is not possible. However, a simple substitution shows that

$$\frac{\|p-\overline{p}\|_G}{\|g\|_{G^{-1}}} = \left|\frac{Q(p)-Q(\overline{p})}{Q(\overline{p})}\right|.$$

In this section we will show how to use this relation to derive a practical convergence criterion. First of all, two lemmas are required.

Lemma (4.3.1) If G is symmetric and positive-definite then

$$y^T G^2 y \leq \|G\| y^T G y.$$

Proof We can assume, without loss of generality, that $G = \text{diag}\{\lambda_i\}$. Then,

$$\begin{aligned} y^T G^2 y &= \sum y_i^2 \lambda_i^2 \\ &\leq \sum y_i^2 \lambda_i \lambda_{\max} \\ &= \lambda_{\max} \sum y_i^2 \lambda_i \\ &= \|G\| y^T G y. \blacksquare \end{aligned}$$

Lemma (4.3.2) If G is symmetric and positive-definite then

$$y^T G y \leq \|G^{-1}\| y^T G^2 y. \blacksquare$$

The proof of this lemma is almost identical to the previous proof and is therefore omitted.

We now move on to our main result.

Theorem (4.3.3) Suppose $Q(p) = \frac{1}{2} p^T G p + p^T g$, where G is symmetric and positive-definite. Let $\overline{p}$ denote the point that minimizes Q. Then, for any p,

$$\begin{aligned} \|Gp+g\|^2 &\leq 2\|G\| \cdot (Q(p)-Q(\overline{p})) \\ (Q(p)-Q(\overline{p})) &\leq \tfrac{1}{2}\|G^{-1}\| \cdot \|Gp+g\|^2. \end{aligned}$$

Proof We will prove here the first result, which relies only on Lemma (4.3.1) above.

The proof of the second result (which relies on Lemma (4.3.2)) is almost identical.

$$\begin{aligned}\|Gp+g\|^2 &= (Gp+g)^T(Gp+g)\\ &= (Gp-G\bar{p})^T(Gp-G\bar{p})\\ &= (p-\bar{p})G^2(p-\bar{p})\\ &\leq \|G\|(p^TGp-2\bar{p}^TGp+\bar{p}^TG\bar{p})\\ &= \|G\|((p^TGp+2p^Tg)-\bar{p}^Tg)\\ &= 2\|G\|(Q(p)-Q(\bar{p})).\blacksquare\end{aligned}$$

The significance of this result is that we can now rewrite Theorem (4.2.4) with $\|r_q\|$ replaced by $(Q(p_q)-Q(\bar{p}))^{1/2}$. One major advantage of using this quantity is that it decreases monotonically during the linear conjugate-gradient iteration.

Clearly, since $\bar{p}$ is unknown during an iteration, we cannot make direct use of this quantity within an algorithm. We require a convergence test which only involves computable quantities, and at the same time maintains a superlinear convergence rate in the outer algorithm. To this end, we will now examine the behavior of the linear conjugate-gradient algorithm more closely. Because the performance of the outer algorithm is now of interest, the superscript $^{(k)}$ is now included in the formulas.

Hestenes [1980] has shown (p. 44) that

$$Q^{(k)}(p_{q+1})-Q^{(k)}(\bar{p}) \leq K\Big[Q^{(k)}(p_q)-Q^{(k)}(\bar{p})\Big],$$

where $K=(\frac{M-m}{M+m})^2$ and m, M are the extreme eigenvalues of $G^{(k)}$. Hence,

$$\begin{aligned}\frac{1-K}{K}\Big(Q^{(k)}(p_{q+1})-Q^{(k)}(\bar{p})\Big) &\leq Q^{(k)}(p_q)-Q^{(k)}(p_{q+1})\\ &\leq Q^{(k)}(p_q)-Q^{(k)}(\bar{p}).\end{aligned}$$

From this we conclude that it is possible to achieve superlinear convergence by insuring that

$$\Big[Q^{(k)}(p_q)-Q^{(k)}(p_{q+1})\Big]^{\frac{1}{2}} = o(\|g^{(k)}\|). \tag{4.3.4}$$

The significance of this result is that all the quantities involved can be computed during the course of the linear conjugate-gradient iteration. Thus, this is a practical way of being able to guarantee superlinear convergence. It is also attractive because it is based on the successive values of the quadratic function minimized by the linear CG algorithm, and not on successive values of the residual. Unfortunately, like the norm of the residual, it does not decrease monotonically during the iteration.

When using the linear conjugate-gradient algorithm to solve $Gp = -g$, we clearly have that $\|p_0\| = \|-g\|$ and that $\|p_\infty\| = \|-G^{-1}g\| \leq \|G^{-1}\| \cdot \|g\|$. This implies that we can rescale (4.3.4) and obtain the following two equivalent convergence tests:

$$[Q^{(k)}(p_q) - Q^{(k)}(p_{q+1})]^{\frac{1}{2}} = o(|Q^{(k)}(p_1)|^{\frac{1}{2}}) \tag{4.3.5}$$

and

$$[Q^{(k)}(p_q) - Q^{(k)}(p_{q+1})]^{\frac{1}{2}} = o(|Q^{(k)}(p_{q+1})|^{\frac{1}{2}}). \tag{4.3.6}$$

These measure the current reduction in the quadratic function against its initial value and its latest value, respectively.

There is one further extension of the convergence results which is of interest when truncated-Newton methods are used in conjunction with the linear algorithm MINRES. This concerns the use of norms which vary from iteration to iteration.

When MINRES is used with preconditioning (see Chapter 5 for details) the progress of the linear iteration is assessed using

$$\frac{\|r^{(k)}\|_{M^{(k)}}}{\|g^{(k)}\|_{M^{(k)}}}, \tag{4.3.7}$$

where $\|\cdot\|_M$ is defined by

$$\|y\|_M^2 = y^T M y$$

and $\{M^{(k)}\}$ is a sequence of positive-definite matrices. From the way in which the MINRES algorithm is derived, it is clear that (4.3.7) has the desired monotonicity property which the original convergence criterion and (4.3.4) lack.

We shall assume that there exists a uniform finite upper bound on the eigenvalues of $\{M^{(k)}\}$. That is, there exists $\lambda_{\max}$ such that

$$\lambda(M^{(k)}) \leq \lambda_{\max} < \infty, \qquad \forall k, \tag{4.3.8}$$

where $\lambda(M^{(k)})$ is any eigenvalue of the matrix $M^{(k)}$. From the way in which the sequence $\{M^{(k)}\}$ is constructed, it will follow that there is a uniform positive lower bound as well. Thus, there exists $\lambda_{\min}$ such that

$$0 < \lambda_{\min} \leq \lambda(M^{(k)}). \tag{4.3.9}$$

Using (4.3.8) and (4.3.9) it is clear that

$$\lambda_{\min}^{\frac{1}{2}}\|y\| \leq \|y\|_{M^{(k)}} \leq \lambda_{\max}^{\frac{1}{2}}\|y\|, \qquad \forall k$$

and hence that

$$\left(\frac{\lambda_{\min}}{\lambda_{\max}}\right)^{\frac{1}{2}} \frac{\|r^{(k)}\|}{\|g^{(k)}\|} \leq \frac{\|r^{(k)}\|_{M^{(k)}}}{\|g^{(k)}\|_{M^{(k)}}} \leq \left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)^{\frac{1}{2}} \frac{\|r^{(k)}\|}{\|g^{(k)}\|}, \qquad \forall k.$$

From this it follows that the ratio (4.3.7) can be used to terminate the linear iteration.

4.4. Practical Forcing Sequences

The forcing sequence in the convergence test can be chosen in accordance with Theorem (4.2.4). This theorem shows how to obtain linear, superlinear, or quadratic convergence by appropriately defining ϕ_k.

If we select $0 < \phi_k \equiv c < 1$ or $0 < c \leq \phi_k \leq d < 1$, then the rate of convergence will be linear. If $\phi_k > 1$, then $p^{(k)} = 0$ is a valid search direction and no step is made; hence we must choose $\phi_k \leq 1$. The reason for choosing $\phi_k < 1$ is to insure that at least one linear iteration will be performed and that a direction other than the steepest-descent direction will be selected if possible. When a preconditioning strategy is employed (see Chapter 5), this condition could be relaxed.

For superlinear convergence, we must have that $\phi_k \to 0$ as $k \to \infty$. Dembo et al. [1980] have suggested using

$$\phi_k = 1/k. \tag{4.4.1}$$

This sequence converges to zero quite slowly, so that the convergence test for the linear iteration is not overly stringent.

Quadratic convergence can be attained if

$$\phi_k \leq \|g^{(k)}\|. \tag{4.4.2}$$

Away from the solution, $\|g^{(k)}\|$ will often be greater than one, so that setting $\phi_k = \|g^{(k)}\|$ may not lead to convergence. This suggests combining (4.4.1) and (4.4.2) to obtain

$$\phi_k = \min\left\{1/k, \|g^{(k)}\|\right\} \tag{4.4.3}$$

as a forcing sequence. Hereafter, we will refer to (4.4.3) as the "standard" forcing sequence.

It may be desirable or necessary to limit the number of linear iterations allowed during each outer iteration. Because the linear inner algorithm converges at a linear

rate, i.e.

$$Q_{q+1} - Q^* \leq K[Q_q - Q^*],$$

then setting an upper bound of L linear iterations leads to a linear rate of convergence for the total algorithm.

Numerical tests of various forcing sequences are conducted in Chapter 7.

4.5. Trust-Region Methods

The quadratic approximation (2.2.2) to the objective function is clearly not valid for all values of p. Up until this point, a line search algorithm has been used to monitor the effectiveness of the approximation and to correct for its deficiencies. Another approach to this problem is to use trust-region methods.

Trust-region methods, like line-search methods, are concerned with minimizing the quadratic approximation (2.2.2). But unlike line search methods, a constraint is added to the subproblem which involves an estimate of the size of the region where (2.2.2) adequately predicts the decrease in value of the objective function F. In exact terms, then, we seek to solve

$$\min_p Q^{(k)}(p) \equiv \min_p F^{(k)} + g^{(k)^T} p + \tfrac{1}{2} p^T G^{(k)} p \tag{4.5.1}$$

subject to

$$\|p\| \leq \delta^{(k)}. \tag{4.5.2}$$

In order to obtain a solution to this problem, the constraint is usually rewritten as $\frac{1}{2}p^Tp \leq \frac{1}{2}(\delta^{(k)})^2$. This transforms (4.5.1) and (4.5.2) into a convex programming problem. The global solution is obtained by finding p and λ such that

$$\begin{aligned} g^{(k)} + G^{(k)}p + \lambda p &= 0 \\ \tfrac{1}{2}p^Tp - \tfrac{1}{2}\delta^{(k)} &\leq 0 \\ \lambda\left(\tfrac{1}{2}p^Tp - \tfrac{1}{2}\delta^{(k)}\right) &= 0 \\ \lambda &\geq 0. \end{aligned} \tag{4.5.3}$$

The problem (4.5.3) is usually solved by computing some estimate of the Lagrange multiplier λ and solving the system of equations

$$\left(G^{(k)} + \lambda I\right)p^{(k)} = -g^{(k)}. \tag{4.5.4}$$

A check is then made to insure that the constraint (4.5.2) is satisfied. If not, a new λ is computed and (4.5.4) is employed once again.

The final step in a trust region iteration involves computing $x^{(k+1)}$ and $\delta^{(k+1)}$. This step is usually based on the value of $\rho^{(k)}$, the ratio between the actual function decrease and the predicted decrease:

$$\rho^{(k)} = \frac{F^{(k)} - F(x^{(k)} + p^{(k)})}{F^{(k)} - Q^{(k)}(x^{(k)} + p^{(k)})}. \tag{4.5.5}$$

Generally, the larger the value of $\rho^{(k)}$, the more adequately the constrained subproblem (4.5.1), (4.5.2) indicates a decrease in the objective function. Thus, if $\rho^{(k)}$ is large, a step is made to the new point $x^{(k+1)} = x^{(k)} + p^{(k)}$. If $\rho^{(k)}$ is especially large, the trust region parameter $\delta^{(k)}$ will be increased, indicating greater confidence in the quadratic approximation. On the other hand, if $\rho^{(k)}$ is small, no step will be made (i.e. $x^{(k+1)} = x^{(k)}$) and $\delta^{(k)}$ will be decreased.

The above is intended as a general outline of trust region methods. More detailed information as well as exact computational formulas can be found, for example, in Vardi [1980] or Hebden [1973].

When the Newton equations (2.2.3) are being solved iteratively, repeated solution of (4.5.4) is impractical. Steihaug [1980] has shown that use of (4.5.4) can be avoided if the constraint (4.5.2) is used to terminate the linear conjugate-gradient algorithm. Steihaug's work was done in the context of truncated quasi-Newton methods (similar to truncated-Newton methods, except that an approximate solution is obtained for the quasi-Newton equations (2.3.3) rather than the Newton equations (2.2.3)), but his ideas are immediately applicable here.

At each iteration of the linear conjugate-gradient algorithm, Steihaug suggests monitoring the length of $\|p_q\|$. This leads to the following formulas and tests:

1. $p_{q+1} = p_q + \alpha_q u_q$
2. If $\|p_{q+1}\| < \delta^{(k)}$ then continue with the algorithm.
3. Otherwise compute $\tau > 0$ such that $\|p_q + \tau u_q\| = \delta^{(k)}$, set $p^{(k)} = p_q + \tau u_q$, and terminate.

Steihaug has managed to show that the resulting algorithm is globally convergent, and is able to prove theorems comparable to (4.2.4) on the actual rates of convergence for various forcing sequences.

Little computational experience has been reported for truncated-Newton algorithms based on trust-region strategies. Although they show obvious promise, they lie outside the scope of this thesis, and will not be considered further. Much of the work used in designing linesearch-type algorithms is directly applicable to the trust region approach.

5 Preconditioning

5.1. Introduction

For every numerical algorithm there is an ideal problem. For Newton's method, the ideal problem is a quadratic function. For the quasi-Newton and conjugate-gradient methods, the ideal problem is a quadratic function with Hessian matrix equal to the identity. More generally, we often think of the class of problems which the algorithm solves well. For Newton's method, this is the set of functions which are "nearly" quadratic. For quasi-Newton and conjugate-gradient methods, it is the set of functions whose Hessians have clustered eigenvalues.

Because most problems are not ideal for an algorithm, it is important to have alternative techniques of modifying the initial problem (without altering its solution) so that it is easier to solve. The general idea is to make the given problem "closer" to the ideal problem. This type of technique is called *preconditioning*.

Preconditioning is such a powerful and general idea that there exist preconditioned versions of almost every known numerical algorithm, both direct and iterative. Direct algorithms often use preconditioning to reduce the error in the computed solution. One common example of this is the use of column scaling in Gaussian elimination (see, for example, Wilkinson [1965], chapter IV). Iterative methods generally use preconditioning to speed up the rate of convergence (although they may also be concerned with the condition of the problem). One of the best-known and best-understood examples of this is the generalized (i.e. preconditioned) linear conjugate-gradient algorithm (Concus, Golub, and O'Leary [1976]). A brief description of preconditioning for the linear-conjugate gradient algorithm can be found in section 3.8.

To give some idea of the versatility of this concept, it is possible to consider Newton, quasi-Newton, and conjugate-gradient algorithms as preconditioned steepest-descent algorithms, with the preconditioning being generated as the algorithm proceeds and being modified at each iteration.

In large problems where it is expensive to compute information, it is important to make as much use as possible of every computed quantity. This generally takes the form of using current information to precondition future iterations.

With truncated-Newton methods, there are two algorithms to be concerned with.

First of all, there is the outer nonlinear iteration. In the basic method, this is just the steepest-descent method. This could be replaced by a conjugate-gradient or a limited-memory quasi-Newton method (using Newton's method would defeat the whole purpose). This idea will be discussed in Section 5.2. During the linear algorithm, matrix-vector products involving the current Hessian matrix are computed. It would be desirable to use these to precondition future non-linear and linear iterations. This is the subject of Sections 3 through 6.

We believe that the range of problems for which a truncated-Newton method will be successful will be extended considerably only when a good direction can be produced in a *small* number of linear conjugate-gradient iterations, and to this end the use of preconditioning is essential.

5.2. Preconditioning with a Non-Linear Algorithm

When using a preconditioned modified Lanczos algorithm to approximately solve the Newton equations (2.2.3), at each iteration it is necessary to solve a system of equations

$$Mz = r$$

involving the preconditioning matrix M. Most non-linear optimization algorithms can be viewed as computing a search direction by solving, possibly implicitly, a system of linear equations

$$Bp = -g,$$

for some matrix B. Thus, by applying the formulas for the non-linear method to the vector r, it is possible to implicitly define a matrix M which can then be used as a preconditioning matrix in the linear algorithm.

Setting $M = I$, i.e. using an unpreconditioned algorithm, corresponds to the steepest-descent method. Another possible preconditioning matrix for this system is an r-step limited-memory quasi-Newton matrix H. As we approach the solution, and F looks more and more like a quadratic function, a small number of quasi-Newton steps can often produce a search direction which is much superior to the steepest-descent direction or to a traditional non-linear conjugate-gradient direction (see Gill and Murray [1979]).

Using a quasi-Newton preconditioning, the vector $-Hg^{(k)}$ will be the first non-trivial member of the sequence $\{p_q\}$ and this direction is far more likely to give a good reduction

in the function than $-g^{(k)}$. Consequently, even if the linear conjugate-gradient algorithm were terminated immediately, a reasonable search direction would have been obtained.

5.3. Diagonal Preconditioning of the Nonlinear Algorithm

Nonlinear minimization algorithms have been found to work more efficiently if the variables are properly scaled. This means that all of the variables are correctly weighted, i.e. that a unit step along the search direction will approximate the minimum of the function in that direction. It also implies that the tolerances for the algorithm have the correct scaling (a factor even for the more scale-invariant algorithms such as Newton's method). One way of achieving this is through a diagonal preconditioning.

If the direction of search is obtained from the quasi-Newton equation (2.3.2) (which is the case when a limited-memory quasi-Newton algorithm is used as a preconditioning strategy), the BFGS formula (2.3.9) may be simplified so that the matrix $B^{(k)}$ does not appear in the rank-two correction:

$$B^{(k+1)} = B^{(k)} + \frac{1}{g^{(k)T}p^{(k)}} g^{(k)}g^{(k)T} + \frac{1}{\alpha^{(k)}y^{(k)T}p^{(k)}} y^{(k)}y^{(k)T}.$$

This result implies that even if the off-diagonal elements of $B^{(k)}$ are unknown, the exact *diagonal* elements can still be recurred. These diagonal elements may be used to precondition the conjugate-gradient method. Let γ_j and ψ_j denote the j^{th} elements of $g^{(k)}$ and $y^{(k)}$ respectively. If $\Delta_{k+1} = \text{diag}(\overline{\delta}_1, \ldots, \overline{\delta}_n)$ and $\Delta_k = \text{diag}(\delta_1, \ldots, \delta_n)$ denote the approximate diagonal Hessians during the $(k+1)^{\text{th}}$ and k^{th} iterations respectively, then

$$\overline{\delta}_j = \delta_j + \frac{1}{g^{(k)T}p^{(k)}} \gamma_j^2 + \frac{1}{\alpha^{(k)}y^{(k)T}p^{(k)}} \psi_j^2.$$

This diagonal preconditioning step involves an approximation to the diagonal of G based on $g^{(k)}$, $p^{(k)}$, and $y^{(k)}$. In the linear iteration of a truncated-Newton method, though, matrix/vector products involving G are computed. It would be desirable to use this more exact second-derivative information to compute Δ, the diagonal preconditioning matrix.

Several methods of computing Δ have been developed and tested. The first two are rank-one and rank-two Quasi-Newton updates which are based on the (false) assumption that G is a diagonal matrix. A third is a BFGS update to the diagonal of the approximate

Hessian. In addition, it is possible to use exact information about the diagonal of the Hessian either to precondition the linear algorithm or to initialize the linear preconditioning. Note, however, that even if matrix-vector products of the form Gv can be found, it may be inconvenient to compute G_{ii}.

At each linear iteration, a computation of the form

$$y = Gs$$

is performed. If the symmetric rank-one Quasi-Newton update is rewritten with B replaced by the diagonal matrix Δ, we obtain

$$\Delta_{i+1} = \Delta_i + \frac{(y - \Delta_i s)(y - \Delta_i s)^T}{(y - \Delta_i s)^T s}.$$

Any off-diagonal terms in the rank-one term are ignored. Notice that no matrix need be stored in order to implement this update.

A similar adaptation can be performed for the BFGS formula. This time the result is

$$\Delta_{i+1} = \Delta_i - \frac{1}{s^T \Delta_i s}(\Delta_i s)(\Delta_i s)^T + \frac{1}{y^T s} y y^T.$$

Again, off-diagonal terms in the rank-one terms are ignored, and no matrix storage is required.

There is a further way in which a diagonal quasi-Newton update can be used to approximate to the diagonal of G. Because the linear conjugate-gradient algorithm is equivalent to the BFGS algorithm (when applied to the same quadratic with $B_0 = I$), it is possible to show that $B_n = G$. Thus, if we were able to update only the diagonals of B, at the end of n steps we would have the exact values for the diagonal elements of G. Unlike the two diagonal updates above, this will be an exact rather than an approximate quasi-Newton update.

To develop this update, we will ignore the nonlinear algorithm for the moment, and concentrate our attention on one instance of the linear conjugate-gradient method. We are attempting to minimize the quadratic function

$$\phi(p) = \tfrac{1}{2} p^T G p + p^T c,$$

and hence

$$g(p) = \phi'(p) = Gp + c = -r(p),$$

where $r(p)$ is the residual at p. The linear conjugate-gradient algorithm is initialized with $p_0 = 0$, and at the q^{th} iteration the next estimate of the solution is computed as

$$p_{q+1} = p_q + \alpha_q u_q,$$

where u_q is the search direction and α_q is the step-length.

The BFGS algorithm computes the (same) search direction using the formula

$$B_q u_q = -g_q, \tag{5.3.1}$$

where $g_q \equiv g(p_q)$. If an exact line-search is used, the step-length for the BFGS algorithm is the same as that for the linear conjugate-gradient algorithm. Under the assumptions that $p_0 = 0$, $B_0 = I$, and that the new approximate Hessian B_{q+1} is computed using

$$B_{q+1} = B_q - \frac{1}{s_q^T B_q s_q}(B_q s_q)(B_q s_q)^T + \frac{1}{y_q^T s_q} y_q y_q^T, \tag{5.3.2}$$

both algorithms compute the same estimates of the solution at every stage.

It is possible to adapt (5.3.2) so that only the diagonals of the update need be computed. Using (5.3.1) and

$$s_q = p_{q+1} - p_q = \alpha_q u_q,$$

we can conclude that

$$B_q s_q = -\alpha_q g_q. \tag{5.3.3}$$

The other important fact is

$$y_q = g_{q+1} - g_q = \alpha_q G u_q. \tag{5.3.4}$$

If we incorporate (5.3.3) and (5.3.4) in (5.3.2), we obtain

$$B_{q+1} = B_q - \frac{1}{u_q^T r_q} r_q r_q^T + \frac{1}{u_q^T (G u_q)} (G u_q)(G u_q)^T. \tag{5.3.5}$$

Using (5.3.5), any individual element of B_q can be individually updated.

When the linear conjugate-gradient algorithm is used directly, (5.3.5) is quite adequate. Unfortunately, problems arise when a linearly preconditioned modified-Lanczos algorithm is used instead. First there is the problem of linear preconditioning. The correspondence between the BFGS and the linear conjugate-gradient methods assumes

that no linear preconditioning is used. This is a very easy problem to surmount, since the linear conjugate-gradient algorithm preconditioned by the matrix M is equivalent to the BFGS algorithm initialized with $B_0 = M$. To see this, replace G by $M^{-\frac{1}{2}}GM^{-\frac{1}{2}}$ in the above derivation.

The other problem concerns scaling. When the linear conjugate-gradient algorithm is implicitly implemented using the Lanczos algorithm, the vectors corresponding to the search direction and the residual are not properly scaled. This scaling does not affect the final term in (5.3.5), since the scaling enters equally into the numerator and the denominator. The other rank-one matrix is affected, however. In our implementation of the algorithm, the correctly scaled residual is available. This leaves only the inner product $u_q^T r_q$. Using the recurrence relation for the search direction u_q, and the fact that the residuals are M-orthogonal, it can be shown that

$$u_q^T r_q = z_q^T r_q,$$

where

$$M^{(k)} z_q = r_q.$$

Since our algorithm computes equally-scaled multiples of z_q and r_q as well as the correctly scaled r_q, it is possible to correctly compute the inner product.

Because the Hessian matrix is not always positive-definite, the modified-Lanczos algorithm alters the subproblem it is solving when it runs across evidence of indefiniteness. The preconditioning scheme is trying to approximate the diagonals of the actual Hessian matrix, and two of the preconditioning algorithms described above have the property of hereditary positive-definiteness, so there is some question as to what should be done when the Hessian matrix is modified. We have chosen to omit the diagonal update whenever the matrix goes indefinite. Since our implementation of the modified-Lanczos algorithm only performs one iteration with the modified matrix before returning to the nonlinear algorithm, very little second-derivative information is wasted using this approach.

There is some theoretical evidence to indicate that, among diagonal preconditionings, this final preconditioning strategy is the most effective. Forsythe and Straus [1955] have shown that if the Hessian matrix G has property A, then the diagonal of G is the optimal diagonal preconditioning. This assumption is valid for many problems arising in partial differential equations. Also, in the general case, van der Sluis [1969] has proven that

preconditioning with the diagonal of G will be nearly optimal, in the sense that the condition number of G preconditioned by its diagonal will be at most n times as large as the condition number of the optimally diagonally preconditioned G. Thus, estimating the diagonal of G using the BFGS formula (1.5) should be effective for all problems.

Using (5.3.5) it is possible to compute any number of subdiagonals in addition to the main diagonal. Because this extension is so straightforward, the details will be omitted here.

5.4. Diagonal Preconditioning with MINRES

In sections 5.2 and 5.3, several methods for diagonally preconditioning a truncated-Newton algorithm were described. The first three (the non-linear preconditioning and the rank-one and rank-two diagonal updates) can immediately be applied to MINRES since they do not rely on any special properties of the underlying linear algorithm. However, a fourth preconditioning (a BFGS update to the diagonal of the approximate Hessian) is dependent on the correspondence between the BFGS quasi-Newton algorithm with exact line searches and the linear conjugate-gradient algorithm. In order to adapt this preconditioning strategy to MINRES, we must analyze the correspondences between MINRES and the linear conjugate-gradient method.

The search directions in MINRES are different to those generated in the linear conjugate-gradient method. Consequently, at first sight we cannot implement the fourth preconditioning technique which relied on the relationship between the search directions for the BFGS algorithm applied to a quadratic function and those for the linear conjugate-gradient algorithm. What we shall show, however, is that from information available in the MINRES algorithm, we can easily generate both the required search directions and the required vectors to update the Hessian approximation.

To this end, we define

$$\overline{W}_q \equiv [w_1, \ldots, w_{q-1}, \overline{w}_q] \equiv V_q Q_q^T,$$

and

$$W_q \equiv [w_1, \ldots, w_q].$$

If the Lanczos process stops with $\beta_{m+1} = 0$, it is then easily verified that

$$GN_m = V_m T_m L_m^{-T} = V_m Q_m^T = W_m. \tag{5.4.1}$$

It is now straightforward to establish the desired correspondences. Using results from Paige and Saunders [1975], we obtain

$$p_q^{CG} - p_q^R = \tau_q (s_q/c_q)^2 n_q \equiv \gamma_q n_q \tag{5.4.2}$$

and

$$\begin{aligned} r_q^{CG} &= r_q^R - \gamma_q w_q \\ &= \beta_1 s_1 s_2 \cdots s_q (s_q w_q - v_{q+1}))/c_q - \gamma_q w_q \\ &= (-\beta_1 s_1 s_2 \cdots s_q v_{q+1})/c_q \equiv \delta_{q+1} v_{q+1}. \end{aligned} \tag{5.4.3}$$

Recall that we are trying to compute

$$B_{q+1} = B_q - \frac{1}{u_q^T r_q} r_q r_q^T + \frac{1}{u_q^T (G u_q)} (G u_q)(G u_q)^T,$$

where the vectors u_q and r_q refer to the search-direction and the residual from the linear conjugate-gradient algorithm. Formula (5.4.3) indicates how to compute r_q for this update. Since

$$p_{q+1} = p_q + \alpha_q u_q$$

and

$$p_{q+1}^R = p_q^R + \tau_q n_q,$$

we can subtract these two equations from each other and use (5.4.2) to obtain

$$\alpha_q u_q = (\gamma_{q+1} + \tau_{q+1}) n_{q+1} - \gamma_q n_q. \tag{5.4.4}$$

Multiplying (5.4.4) by G and using (5.4.1) we obtain

$$\alpha_q G u_q = (\gamma_{q+1} + \tau_{q+1}) w_{q+1} - \gamma_q w_q. \tag{5.4.5}$$

Consequently, the vector Gu_q need not be calculated directly. This is of particular significance in the non-linear algorithm when G may be unknown. Thus, we are able to compute a scaled version of the conjugate-gradient search direction. Since the final term in the BFGS update is scale-invariant, we can use (5.4.4) and (5.4.5) in order to compute it. This is not true of the first term, but α_q can be computed using

$$\alpha_q^{-1} = \frac{r_q^T (M^{-1} r_q)}{(\alpha_q u_q)^T (\alpha_q G u_q)} = \frac{\delta_{q+1}^2 v_{q+1}^T (M^{-1} v_{q+1})}{(\alpha_q u_q)^T (\alpha_q G u_q)} = \frac{\delta_{q+1}^2}{(\alpha_q u_q)^T (\alpha_q G u_q)},$$

where M is the preconditioning matrix for the Lanczos algorithm. Combining all of these

results, we obtain the desired formula for the BFGS update:

$$B_{q+1} = B_q - \frac{\alpha_q \delta_{q+1}}{(\alpha_q u_q)^T r_q} v_{q+1} v_{q+1}^T + \frac{1}{(\alpha_q u_q)^T (\alpha_q G u_q)} (\alpha_q G u_q)(\alpha_q G u_q)^T.$$

5.5. Tridiagonal Preconditiong

The linear conjugate-gradient algorithm transforms the Hessian matrix G according to the formula

$$GR_q = V_q T_q + \beta_{q+1} v_{q+1} e_q^T.$$

Thus

$$R_q^T G R_q = T_q.$$

This suggests the use of the preconditioning matrix

$$V_q T_q V_q^T \equiv M_q.$$

The matrix M_q is rank deficient and consequently cannot be used directly as a preconditioning matrix. By extending the definitions of V_q and T_q, we can construct a preconditioning matrix utilizing the information in M_q.

In order to extend V_q, we form its QR factorization (using, for example, Householder transformation (see Wilkinson [1965], pp. 290–299)):

$$V_q = QR,$$

where Q is an $n \times n$ orthogonal matrix; R is of the form

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$

and R_1 is a $k \times k$ upper-triangular matrix. If we partition Q conformally to R:

$$Q = (Q_1 \mid Q_2),$$

the columns of Q_1 span the same space the columns of V_q, and the columns of Q_2 span its orthogonal complement. To complete the extension of V_q to the whole space, we define

$$\tilde{V}_q = Q \begin{pmatrix} R_1 & 0 \\ 0 & I \end{pmatrix}.$$

To extend T_q, we exploit the convergence theory for the Lanczos algorithm. Parlett [1980] has shown that the eigenvalues of T_q tend toward the extreme eigenvalues of G. It is natural, then, to define

$$\tilde{T}_q = \begin{pmatrix} T_q & 0 \\ 0 & \gamma I \end{pmatrix},$$

where

$$e_1 \equiv \lambda_{\min}(T_q) \leq \gamma \leq \lambda_{\max}(T_q) \equiv e_n.$$

Some possible choices for γ are

$$\gamma = \tfrac{1}{2}(e_1 + e_n)$$

and

$$\gamma = (e_1 \cdot e_n)^{\frac{1}{2}}.$$

The full preconditioning is then

$$\tilde{M}_q \equiv Q\tilde{R}\tilde{T}_q\tilde{R}^T Q^T.$$

A similar tridiagonal preconditioning can be produced by using the approximation

$$G \approx \tilde{R}_q^{-T}\tilde{T}_q\tilde{R}_q^{-1},$$

where $\tilde{R}_q$ refers to the extension of R_q to the whole space (as in the definition of $\tilde{V}_q$).

5.6. Approximating the Product of the Tridiagonal Preconditionings

Although at the first iteration the tridiagonal matrix from the Lanczos algorithm has eigenvalues which approximate the extreme eigenvalues of G, at subsequent iterations the Lanczos algorithm is being applied to a preconditioned version of G whose extreme eigenvalues may bear little relation to those of G itself—in fact, this is the intent of the preconditioning strategy. One attempt to surmount this problem involves computing the product of the previous preconditioning matrices $\tilde{M}_i$, and using this product as the preconditioning at the next iteration. Because of storage limitations, this product cannot be computed exactly, and an approximation to it must be used. That approximation is the topic of this section.

Suppose we have already computed

$$\overline{M}_{k-1} \approx \prod_{i=1}^{k-1} \tilde{M}_i$$

in the factored form

$$\overline{M}_{k-1} = (V_{k-1} \mid V_{k-1}^{\perp}) \begin{pmatrix} \tilde{L}_{k-1}\tilde{L}_{k-1}^T & 0 \\ 0 & \overline{\gamma}_{k-1}^2 I \end{pmatrix} (V_{k-1} \mid V_{k-1}^{\perp})^T.$$

Also assume that the current preconditioning matrix $\tilde{M}_k$ is available in the form

$$\tilde{M}_k = (V_k \mid V_k^{\perp}) \begin{pmatrix} L_k L_k^T & 0 \\ 0 & \gamma_k^2 I \end{pmatrix} (V_k \mid V_k^{\perp})^T.$$

Here, L_k is lower-bidiagonal and $\tilde{L}_{k-1}$ is lower triangular. Then, by applying $\tilde{M}_k^{\frac{1}{2}}$ on the left and right to $\overline{M}_{k-1}$, we obtain an approximation to $\overline{M}_k$:

$$\overline{M}_k \approx (V_k \mid V_k^{\perp}) \begin{pmatrix} L_k & 0 \\ 0 & \gamma_k I \end{pmatrix} \overline{M}_{k-1} \begin{pmatrix} L_k^T & 0 \\ 0 & \gamma_k I \end{pmatrix} (V_k \mid V_k^{\perp})^T.$$

If we treat the central factors in this product as block 2×2 matrices, compute their product, and ignore off-diagonal terms, we obtain

$$\begin{aligned} \overline{M}_k &\approx (V_k \mid V_k^{\perp}) \left(\begin{pmatrix} L_k & 0 \\ 0 & 0 \end{pmatrix} (V_{k-1}\tilde{L}_{k-1}\tilde{L}_{k-1}^T V_{k-1}^T) \begin{pmatrix} L_k^T & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & \gamma_k^2 \overline{\gamma}_{k-1}^2 I \end{pmatrix} \right) (V_k \mid V_k^{\perp})^T \\ &\equiv (V_k \mid V_k^{\perp}) \begin{pmatrix} \tilde{L}_k \tilde{L}_k^T & 0 \\ 0 & \overline{\gamma}_k^2 I \end{pmatrix} (V_k \mid V_k^{\perp})^T \end{aligned},$$

where $\tilde{L}_k$ is lower triangular and $\overline{\gamma}_k = \gamma_k \overline{\gamma}_{k-1}$. $\tilde{L}_k$ is obtained by doing a Cholesky factorization of the first matrix in the sum above. This is the desired approximation to the product of the preconditioning matrices.

6 Extensions to Other Problems

6.1. Introduction

So far, we have been concerned with the solution of unconstrained minimization problems where few assumptions have been made about the form of the objective function F. It is very common to encounter problems where auxiliary conditions, called constraints, are placed on the independent variables x. Another common problem is the minimization of functions which can be represented as the sum of the squares of other functions. Such problems are often referred to as least-squares problems.

In this chapter, we show how truncated-Newton methods can be adapted to solve problems of both these types. In sections 6.2–6.4 we discuss constrained problems, and in section 6.5 a treatment of least-squares problems is given. Since successful methods already exist for dense problems of these types, we are especially concerned with large sparse problems. In particular, currently it is difficult to solve constrained problems where the number of variables n, the number of constraints t, and their difference $n - t$ are all large. Truncated-Newton methods provide some hope in this case.

6.2. Constrained Minimization Problems

The most general constrained optimization problem can be expressed in the form

$$\min_x F(x) \tag{6.2.1}$$

subject to the conditions

$$c_i(x) \geq 0, \qquad i = 1, \ldots, m. \tag{6.2.2}$$

Here, $F(x)$ and $c_i(x)$ are functions mapping from $\Re^n \rightarrow \Re$.

Problems of this type are often further classified by the form of the constraints (6.2.2). The major division is made between linear and non-linear constraints. The constraints are also further divided into groups of equality and inequality constraints.

The form of the constraints can strongly affect the way in which the problem (6.2.1), (6.2.2) is solved. Methods for problems with linear equality constraints will be discussed in section 6.3, and linear inequality constraints in section 6.4. Methods for non-linearly constrained problems are still an active research area. It is not yet clear how best to apply Newton's method to such problems when the number of variables is large. For

this reason, and because the technical details of such methods can strongly affect the development of the relevant theory, we will only consider the application of truncated-Newton methods to linearly constrained problems. Much of what we describe will be relevant to the non-linear case.

6.3. Problems with Linear Equality Constraints

The problem we are concerned with in this section (denoted hereafter as ECP) is usually written in the form

$$\min_x F(x) \tag{6.3.1}$$

subject to

$$Ax = b, \tag{6.3.2}$$

where A is a $t \times n$ matrix.

The set of constraints (6.3.2) restricts the set of "feasible" points, i.e. the set of points which will be considered when solving (6.3.1). As usual,we denote the solution to ECP by x^*. If $\bar{x}$ is any other feasible point, then

$$\begin{aligned} A(\Delta x) &\equiv A(x^* - \bar{x}) \\ &= Ax^* - A\bar{x} \\ &= b - b = 0. \end{aligned}$$

Also, if p is any vector satisfying

$$Ap = 0,$$

then

$$\begin{aligned} A(x^* + p) &= Ax^* + Ap \\ &= b + 0 \\ &= b, \end{aligned}$$

so $x^* + p$ is also feasible. Thus, all feasible steps from the point x^* are orthogonal to the rows of A (or, equivalently, to the columns of A^T).

Let Z denote a matrix whose columns form a basis for the null space of A^T, i.e. $AZ = 0$. Then any feasible point must be of the form $x^* + Zp_z$ for some p_z. If we examine the Taylor series for F around the point x^*, we find that

$$F(x^* + \epsilon p) = F(x^* + \epsilon Z p_z) = F(x^*) + \epsilon p_z^T Z^T g(x^*) + \frac{\epsilon^2}{2} p_z^T Z^T G(x^*) Z p_z + \cdots. \tag{6.3.3}$$

Clearly, if x^* is the minimizing point for the constrained problem, then

$$Z^T g(x^*) = 0. \tag{6.3.4}$$

The quantity $Z^T g$ will be referred to as the *projected gradient.*

The condition (6.3.4) implies that $g(x^*)$ lies in the range of A^T:

$$\begin{aligned} 0 &= Z^T g(x^*) \\ &= Z^T(A^T g_A + Z g_z) \\ &= Z^T Z g_z. \end{aligned}$$

Hence, $g_z = 0$. The coefficients of the vector g_A are denoted $\lambda_1^*, \ldots, \lambda_t^*$ and are called the *Lagrange multipliers* at x^*.

Using (6.3.4) and (6.3.3) we find that

$$F(x^* + p) = F(x^*) + \frac{\epsilon^2}{2} p_z^T Z^T G(x^*) Z p_z + \cdots,$$

so that the matrix $Z^T G Z$ (the *projected Hessian*) must be positive semi-definite at x^*.

Thus we have obtained first- and second-order necessary conditions for a solution to problem ECP. Sufficient conditions, derived in the same way, are (if x^* is a feasible point):

(i) $Z^T g(x^*) = 0 \qquad (g(x^*) = A\lambda^*)$.

(ii) $Z^T G(x^*) Z$ is positive definite.

We now move on to derive methods for solving ECP. To do this, we return to the Taylor expansion (6.3.3). The function F is expanded about an arbitrary feasible point x in the direction $Z p_z$ with $\epsilon = 1$, and the series is truncated after the quadratic term. We obtain

$$F(x + Z p_z) = F(x) + p_z^T Z^T g + \tfrac{1}{2} p_z^T Z^T G Z p_z.$$

Setting $F(x + Z p_z) = 0$, we obtain that the step to the minimum of this quadratic is found by solving the *projected-Newton equations*:

$$G_z p = -g_z, \tag{6.3.5}$$

where

$$\begin{aligned} G_z &= Z^T G Z, \\ g_z &= Z^T g. \end{aligned}$$

Because of the similarity between (6.3.5) and the Newton equations (2.2.3), it is easy to derive a modified-Newton algorithm for solving ECP by specifying that $G^{(k)}$ and $g^{(k)}$ be replaced by $G_z^{(k)}$ and $g_z^{(k)}$ in all the relevant formulas.

Truncated-Newton methods can be extended just as easily. All that is required is a subroutine to compute products of the form

$$y = G_z^{(k)} v$$

for any v. The fact that a projected Hessian is being used is irrelevant to the algorithm.

In the above discussion, we have not stated how to obtain Z, the basis for the null-space of A^T. In the dense case, it is possible to derive Z from a LQ factorization of A. (We will assume that A has full row rank.) There exists an orthogonal matrix Q which, when applied to A on the right, yields an lower-triangular matrix:

$$AQ = \overline{L} = (L \quad 0), \qquad \text{i.e. } A = Q^T \overline{L},$$

where L is a $t \times t$ lower-triangular matrix. We partition Q conformally to $\overline{L}$:

$$Q = (Y \; Z),$$

so that

$$AQ = (AY \; AZ) = (L \quad 0).$$

Thus, the last $(n - t)$ columns of Q are orthogonal to the rows of A, and Z is the desired basis for the null space of A^T. The LQ factorization above can be computed using elementary orthogonal transformations (such as Householder reflections or Givens rotations).

In the sparse case, it is usually preferable to use a technique known as *variable reduction* to form Z. We partition the constraint matrix A in the form

$$A = (T \; U),$$

where T is a $t \times t$ non-singular matrix. For simplicity, we have assumed that T corresponds to the first t columns of A. With this partitioning of A, a matrix Z orthogonal to the rows of A can be defined as

$$Z = \begin{pmatrix} -T^{-1}U \\ I \end{pmatrix}.$$

If Z is defined in this way, it need not be explicitly formed. We need only be able to solve systems of equations involving T and T^T, so that all that is required is a factorization

of T. Note that the matrix T^{-1} is not obtained explicitly. To compute the product Z times a vector, we perform back-substitution using the factors of T. This allows us to exploit sparsity in the constraint matrix, although in general the condition number of the matrix Z obtained from variable reduction will be larger than for the matrix Z obtained from an LQ factorization (in that case, we will have $\kappa(Z) = 1$).

6.4. Linear Inequality Constraints

6.4.1. Theory

The problem we are considering in this section (denoted by ICP) will be posed in the form

$$\min_x F(x) \tag{6.4.1}$$

subject to

$$\begin{aligned} Ax &= b, \\ l \leq x &\leq u, \end{aligned} \tag{6.4.2}$$

where A is an $m \times n$ matrix. Notice that the only inequality constraints are just simple bounds on the variables. General inequality constraints are converted to this form by the introduction of slack variables. This problem is considerably more complex than the equality-constraint problem ECP since we do not know in advance which bounds (if any) will be exactly satisfied as equalities at the solution.

If $\hat{A}$ were the set of constraints active at the solution, then the solution of ICP would also be the solution of the equality-constraint problem $\min F(x)$ subject to $\hat{A}x = \hat{b}$. This suggests applying techniques for the equality-constraint case to ICP. We will obtain the solution to problem ICP by solving a sequence of minimization problems subject to linear equality constraints. The objective function (6.4.1) remains the same in all of these subproblems, but the constraint matrix is modified to reflect the current assumptions about which bounds are satisfied as equalities at the solution. We will refer to the current set of constraints as the *working set* and will assume that they correspond to the equation $\hat{A}x = \hat{b}$. As before, Z will be used to denote a matrix satisfying $\hat{A}Z = 0$.

The working set will contain a subset of the original problem constraints, and will attempt to predict the correct active set. Since the prediction of the active set could be wrong, an active set method must also include procedures for testing whether the current prediction is correct and altering it if not. An essential feature of the active set methods

considered here is that all iterates are feasible.

Following the development in Murtagh and Saunders [1978], the matrix A is partitioned as

$$A = (B\ S\ N) \tag{6.4.3}$$

where B is a square $m \times m$ non-singular matrix, N is $m \times r$, and S is $m \times (n - m - r)$. B is called the *basis* matrix and its columns correspond to the *basic* variables. The columns of N correspond to the *nonbasic* variables, i.e. those variables which are equal to one of their bounds. The columns of S correspond to the remaining variables, which are called *superbasic*. The number of superbasic variables indicates the number of degrees of freedom remaining in the minimization. In the important special case of linear programming where F is a linear function, the matrix S is null. With the partitioning (6.4.3) of the matrix A, we can write the constraints for the subproblem in the form

$$\hat{A}x = \begin{pmatrix} B & S & N \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} x_B \\ x_S \\ x_N \end{pmatrix} = \begin{pmatrix} b \\ b_N \end{pmatrix} = \hat{b}, \tag{6.4.4}$$

where the components of b_N are taken from either l or u, depending on whether the lower or upper bound is binding.

We expand the function F in a Taylor series about some feasible point x:

$$F(x+p) = F(x) + g(x)^T p + \tfrac{1}{2} p^T G(x) p + \cdots. \tag{6.4.5}$$

If $F(x)$ were a quadratic function, then G would be a constant matrix, and there would be no higher-order terms in this expansion. In this case, we could obtain a constrained stationary point at $x + p$ by insisting that

$$\begin{pmatrix} B & S & N \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} p_B \\ p_S \\ p_N \end{pmatrix} = 0, \tag{6.4.6}$$

i.e. the step remains on the surface given by the intersection of the active constraints; also

$$\begin{pmatrix} g_B \\ g_S \\ g_N \end{pmatrix} + G \begin{pmatrix} p_B \\ p_S \\ p_N \end{pmatrix} = \begin{pmatrix} B^T & 0 \\ S^T & 0 \\ N^T & I \end{pmatrix} \begin{pmatrix} \mu \\ \lambda \end{pmatrix}, \tag{6.4.7}$$

i.e. the gradient at $x + p$ is expressible as a linear combination of the active constraint normals. These two conditions correspond to the conditions $p = Zp_z$ and (6.3.4) in the previous section.

For a more general function $F(x)$, the step p may not lead directly to a stationary point, but (6.4.6) and (6.4.7) can be used to determine a feasible descent direction. From (6.4.6) we have $p_N = 0$ and $p_B = -Wp_S$, where $W = B^{-1}S$. Thus,

$$p = \begin{pmatrix} -W \\ I \\ 0 \end{pmatrix} p_S.$$

Notice the correspondence between this matrix and the matrix Z computed using the variable-reduction method in the previous section. As before, we do not explicitly compute B^{-1}, but instead compute some factorization of this matrix. Back-substitution is then used to compute the necessary products of W times a vector. This matrix is indeed orthogonal to the working set matrix $\hat{A}$. The Lagrange multipliers $(\mu\ \lambda)^T$ can be computed using the equations

$$B^T\mu = g_B + (I \quad 0 \quad 0)G\begin{pmatrix} -W \\ I \\ 0 \end{pmatrix} p_S$$

and

$$\lambda = g_N - N^T\mu + (0 \quad 0 \quad I)G\begin{pmatrix} -W \\ I \\ 0 \end{pmatrix} p_S.$$

When $p_S = 0$, these equations reduce to

$$\mu = B^{-T}g_B$$
$$\lambda = g_N - N^T\mu.$$

The Lagrange multipliers can be used to modify the working set of constraints. For example, suppose that bound i is fixed at its lower endpoint, i.e. we are assuming that $x_i = l_i$. If the Lagrange multiplier λ_i corresponding to this bound is negative, then the objective function F will decrease locally if x_i is allowed to increase in value. This indicates that bound i could be dropped from the working set. A similar situation exists for upper bounds, but there the Lagrange multiplier should be positive if the bound is to be relaxed.

Using the Taylor series (6.4.5) and equation (6.4.6), we obtain that a first-order condition for x^* to solve ICP is that $Z^Tg(x^*) = 0$. This is the same condition as for the equality-constraint problem. We are assuming, of course, that x^* is a feasible point and that $\hat{A}x^* = \hat{b}$.

The Taylor expansion (6.4.5) also leads us to a second-order condition for a solution

to ICP. Because the projected gradient is zero, we obtain directly that

$$G_z = Z^T G Z$$

must be positive semi-definite at x^*.

Sufficient conditions for a minimum are slightly more complex, since the Lagrange multiplier for an active constraint may be zero. Also, complications can arise because a variable could be fixed at either a lower or an upper bound. For simplicity, we assume that all active bounds are lower bounds. Results for upper bounds are obtained by changing $>$ to $<$ below. Keeping this in mind, we find that x^* is a solution to ICP if:

(i) $Ax^* = b$, $l \leq x^* \leq u$ and $\hat{A}^T x^* = \hat{b}$.

(ii) $Z^T g(x^*) = 0$ (where $\hat{A}Z = 0$).

(iii) $\lambda^* \geq 0$ (where λ is obtained from (6.4.7)).

(iv) $Z^T G(x^*) Z$ is positive definite.

(v) If $\lambda_i^* = 0$, then $p^T G(x^*) p > 0$ for all p such that $p_{N_i} > 0$.

Assuming that an initial feasible point is available, the general structure of an working-set algorithm can be summarized as follows:

(6.4.6) Working-set Algorithm

W1. Let $x^{(k)}$ be the current point. We assume that $x^{(k)}$ is feasible and that $\hat{A}$ is the matrix of constraints active at $x^{(k)}$.

W2. Check if $x^{(k)}$ is the solution of the equality-constraint problem. If not, go to step **W5**.

W3. Calculate $\lambda = g_N - N^T \mu$. If λ satisfies the second-order sufficient conditions for a minimum, then x^* is the solution to ICP. Terminate the algorithm.

W4. If $\lambda_i < 0$ for some variable x_{N_i} at its lower bound (or $\lambda > 0$ for some x_{N_i} at its upper bound), compute a direction $p^{(k)}$ such that ${g^{(k)}}^T p^{(k)} < 0$, $p_{N_i}^{(k)} > 0$ ($p_{N_i}^{(k)} < 0$), and $p_{N_j}^{(k)} = 0$ for $i \neq j$. For such a $p^{(k)}$, the i-th bound becomes inactive and is deleted from the active set. Go to step **W6**.

W5. ($g^{(k)} \neq \hat{A}\lambda$) Construct a direction $p^{(k)}$ such that $\hat{A}^T p^{(k)} = 0$, ${g^{(k)}}^T p^{(k)} < 0$ (i.e. a descent direction for the equality-constraint problem). This can be done by solving the projected-Newton equations for the

equality-constraint problem.

W6. (line search) Normally, the line search would be based solely on a "sufficient decrease" in the objective function F. In this context, it is possible to run into a formerly inactive bound while searching along $p^{(k)}$. If a new constraint is encountered during the linesearch, it is then added to the active set. Go to step **W1**.

Because there is possibly some choice in step **W4** as to which constraint to drop from the active set, various active-set strategies have been suggested for solving ICP. For this thesis, where we are concerned with the application of truncated-Newton methods, the details of the strategy are not important.

6.4.2. The Application of Truncated-Newton Methods to Inequality-Constraint Problems

Since the algorithm described in the previous section computes a search direction p by solving a set of projected-Newton equations, it may appear that truncated-Newton methods can be applied directly for the solution of ICP. If no preconditioning is used, this is indeed true. The construction of $p^{(k)}$ in step **W5** is then a local problem involving the (approximate) solution of a set of linear equations.

However, when preconditioning is a part of the algorithm, certain complications arise. With equality constraints, the projection matrix Z remains constant; but with inequality constraints, the projection matrix, and hence the structure and size of the projected-Newton equations (6.3.5), can change from iteration to iteration as the active set changes. In this section, we describe how to modify the preconditioning matrix to reflect these changes.

In order to simplify the discussion, we will assume (without loss of generality) that bounds are added or deleted one at a time. We will also assume that a diagonal preconditioning is being used. More complex preconditionings can be used, and it is straightforward to adapt the following discussion to the more general case. In fact, the ideas here are based on the presentation in Gill and Murray [1973b] where a (full) quasi-Newton approximation to the Hessian is being modified.

Deleting a bound corresponds to deleting a column (say the last) from $\hat{A}$. This implies that a column must be added to Z:

$$\overline{Z} = (Z \mid z).$$

Then

$$\overline{Z}^T G \overline{Z} = \begin{pmatrix} Z^T G Z & Z^T G z \\ z^T G Z & z^T G z \end{pmatrix}.$$

(For large-scale problems, this matrix would never be explicitly computed; we use it here only as a theoretical tool.) Let D be our diagonal preconditioning corresponding to $Z^T G Z$. A natural choice for $\overline{D}$, the new preconditioning, is thus

$$\overline{D} = \begin{pmatrix} D & 0 \\ 0 & \alpha \end{pmatrix},$$

where $\alpha = z^T G z$ (or $\alpha = 1$ if this quantity is expensive to compute).

Adding a bound is a slightly more difficult problem. This corresponds to reducing the size of Z by one column. If we are deleting the q-th superbasic variable, then we just delete the q-th diagonal element of D to obtain $\overline{D}$. Deletion of a basic variable can be achieved by interchanging the basic variable with a superbasic variable, and then deleting the new superbasic column as indicated.

The interchange of the p-th basic variable with the q-th superbasic variable can be described by an equation of the form

$$\overline{Z} = Z(I + e_q v^T),$$

where e_q is the q-th unit vector and v is defined by the equations

$$\begin{aligned} B^T \pi_p &= e_p, \\ y &= S^T \pi_p, \\ y_q &= y^T e_q, \\ v &= \frac{-1}{y_q}(y + e_q). \end{aligned}$$

These quantities are easily computed.

We would now like to approximate the diagonal $\overline{D}$ of

$$\overline{Z}^T G \overline{Z} = (I + e_q v^T)^T Z^T G Z (I + e_q v^T)$$

given Z and an approximation D to the diagonal of $Z^T G Z$. The formula for the new diagonal element $\overline{d}_i$ is

$$\overline{d}_i = d_i + 2 v_i z_q^T G z_i + z_q^T G z_q v_i^2,$$

where z_i is the i-th column of Z. Unless the values $\{z_q^T G z_i\}$ are inexpensive to compute, updating D in this manner will not be feasible. As a result, we suggest simply deleting the q-th column of D; there is little justification for applying the transformation $(I + e_q v^T)$ directly to D.

Further details concerning the treatment of constraint matrices for large problems can be found in Murtagh and Saunders [1978].

6.5. Least-Squares Problems

Least-squares problems are concerned with finding a point x^* which minimizes the sum of squares of nonlinear functions

$$F(x) = \sum_{i=1}^{m} [f_i(x)]^2, \qquad x \in \Re^n, \quad m \geq n. \tag{6.5.1}$$

Such problems can be solved using the minimization algorithms described in the previous chapters, but the special form of the function F suggests the use of more specialized techniques.

The gradient vector $g(x)$ and Hessian matrix $G(x)$ of $F(x)$ are given by $2J(x)^T f(x)$ and $2(J(x)^T J(x) + B(x))$ respectively, where $J(x)$ is the $m \times n$ Jacobian matrix of $f(x)$ whose i-th row is $\nabla f_i(x) = (\partial f_i/\partial x_1, \partial f_i/\partial x_2, \ldots, \partial f_i/\partial x_n)$, $B(x) = \sum_{i=1}^{m} f_i(x) G_i(x)$ and $G_i(x)$ is the Hessian matrix of $f_i(x)$. ($F(x)$ is assumed to be twice continuously differentiable, although the methods discussed in this section will often work when this condition does not hold.) The restriction that m is greater than or equal to n serves only to simplify the notation.

If Newton's method is applied to the solution of (6.5.1), the special form of the Hessian matrix and gradient vector leads to the following set of linear equations for the Newton direction

$$(J(x^{(k)})^T J(x^{(k)}) + B(x^{(k)}))p_N^{(k)} = -J(x^{(k)})^T f(x^{(k)}). \tag{6.5.2}$$

The Gauss-Newton method was the first designed to exploit the special structure of the Hessian matrix and gradient vector which occurs in least-squares problems. The method computes the direction of search as the solution to

$$J(x^{(k)})^T J(x^{(k)}) p_{GN}^{(k)} = -J(x^{(k)})^T f(x^{(k)}). \tag{6.5.3}$$

These equations are obtained by neglecting the second-derivative matrix $B(x^{(k)})$ in (6.5.2). The Gauss-Newton method is intended for problems where $\|B(x)\|$ is small compared to $\|J(x)^T J(x)\|$, such as the so-called "small-residual problem" where $f(x) \to 0$ as $x \to x^*$. For these problems the Gauss-Newton method will ultimately converge at the same rate as Newton's method, despite the fact that only first-derivative information is used. We will concentrate on that case here.

Truncated-Newton methods can be applied directly to the solution of the equations (6.5.3) (or even (6.5.2) if the second derivative information in $B(x)$ is available). The fact that the Hessian matrix and gradient are of a special form is irrelevant to the truncated-Newton algorithm.

If we assume that the matrix J is of full rank, then the system of equations (6.5.3) will have a positive-definite coefficient matrix. Thus, unlike when we were solving more general optimization problems, it is possible to use the regular linear conjugate-gradient algorithm to approximately solve (6.5.3). It is possible to use the algorithm described in section 2.4, if we set $A = J^T J$. However, because of the factored form of the coefficient matrix in (6.5.3), and because we would like to precondition the linear algorithm, the following set of formulas is to be preferred:

Given p_0. Set $s_0 = f - Jp_0$, $r_0 = J^T s_0$.

For $q = 0, 1, \ldots$

$$
\begin{aligned}
z_q &= M^{-1} r_q \\
u_q &= z_q + \beta_q u_{q-1} \\
\beta_q &= z_q^T r_q / z_{q-1}^T r_{q-1} \\
\beta_0 &= 0 \\
v_q &= J u_q \\
x_{q+1} &= x_q + \alpha_q u_q \\
\alpha_q &= z_q^T r_q / v_q^T v_q \\
s_{q+1} &= s_q - \alpha_q v_q \\
r_{q+1} &= J^T s_{q+1}
\end{aligned}
$$

Next q.

Here, M is some approximation to $J^T J$.

The only remaining problem is how to generate the preconditioning matrix M. When the matrix J is available, then we would always use $\text{diag}\{J^T J\}$. Otherwise, we could use one of the diagonal preconditionings described in Chapter 5. They require a pair of vectors (u, Gu), or in this case $(u, J^T J u)$. When the linear algorithm is programmed

using the formulas above, the vector $J^T J u$ is not a natural by-product of the algorithm. However, if we compute the difference between the successive residuals

$$\begin{aligned} r_{q+1} - r_q &= J^T(s_{q+1} - s_q) \\ &= -\alpha_q J^T J u_q, \end{aligned}$$

we are able to obtain the desired vector.

When the derivatives of the functions f_i are available, it is also possible to solve the full Newton equations (6.5.2) using a linear conjugate-gradient algorithm. If the matrix J were sparse, then this algorithm could be preconditioned using $J^T J$. Otherwise, the diagonal of $J^T J$ or an approximation to it could be used.

7 Numerical Results

7.1. Introduction

In this Chapter we discuss the numerical behavior of several of the methods discussed earlier. It was not feasible to test every combination of techniques that has been described, but we have attempted to ascertain through selective testing the most promising version of a truncated-Newton algorithm for general usage. The method used to compare algorithms consists of applying them to a set of test problems. We do not claim that this is a completely satisfactory means of comparison, but we believe that, if the test problems are selected carefully, the evidence obtained can be a valuable aid in the selection of the best algorithm.

This method of testing has many drawbacks. One difficulty is the volume of data that subsequently needs analyzing. We have displayed the raw data together with an aggregation of the results. Too much emphasis, however, should not be placed on the aggregated numbers since they are unduly weighted by the more difficult problems. An alternative form of display is to enter as 1 the best result and have all other entries be their multiple of this result. The drawback to this method is in our view more serious since greater emphasis is then placed on problems that are easily solved.

The popularity and success of battery testing is largely due to the fact that for many algorithms the differences in results are so large as to leave little doubt as to the correct conclusion. It is also a useful technique for demonstrating that an algorithm is poor. The converse, however, is not always true. If an algorithm fails where others easily succeed it demonstrates a flaw in that algorithm. If an algorithm is simply a little slower or faster then this could merely be due to the luck of the draw.

7.2 The assessment criterion

All optimization software requires a criterion for terminating the computation of the sequence $\{x^{(k)}\}$. Ideally, if we wish to measure the comparative efficiency of routines we should set the same termination criterion in all the routines tested and then compute the cost of a minimization, in terms of the number of function evaluations for instance. However, there is no universal agreement on what is the best termination criterion and a different criterion used by another researcher may result in a wide variation in the

accuracy of the answer obtained. The question remains, therefore, as to the point at which we should assess the efficiency of the various methods. The assessment criterion used here is to take the first point $x^{(k)}$ for which

$$F^{(k)} - F(x^*) < \tau(1 + |F(x^*)|), \tag{7.2.1}$$

where τ is a scalar. Some authors have argued against the use of (7.2.1) because it includes $F(x^*)$, which is unknown on real problems. We believe that such authors are confusing an *assessment criterion*, where the use of $F(x^*)$ is legitimate, with a *termination criterion*, where it is not.

If the criterion (7.2.1) is to give a realistic *assessment* of the performance of an algorithm, the choice of τ must give a point $x^{(k)}$ which is close to a final estimate of x^* obtained with a realistic *termination* criterion. The relative performance of algorithms with superlinear convergence is almost invariant with the choice of τ and a very small value can be used. For example, on an IBM 370/168, where the function can be computed to approximately fifteen decimal places in double precision, a reasonable choice of τ is 10^{-10}. However, for conjugate-gradient type methods, which exhibit a linear rate of convergence, the performance can vary widely with the choice of τ. It is not unusual for the number of function evaluations to be three times greater for $\tau = 10^{-10}$ than for $\tau = 10^{-5}$. In this case *it is important that a moderate termination criterion be used.* In all the tests carried out for this study, τ was set at 10^{-5}.

7.3 The algorithms tested

The results of this chapter, in addition to exhibiting the performance of a variety of truncated-Newton algorithms, illustrate the numerical behaviour of three algorithms for general unconstrained minimization. These are:

1. Algorithm PLMA

Diagonally preconditioned two-step BFGS formula with accumulated step (see Gill and Murray [1979]).

2. Algorithm MNM

A modified Newton method using first and second derivatives (see Gill and Murray [1974a]).

3. Algorithm QNM

A quasi-Newton method using the full $n \times n$ BFGS update of the approximate Hessian Matrix (see Gill and Murray [1972]).

The use of these accepted and widely-tested algorithms gives us an objective test of the overall effectiveness of our truncated-Newton methods.

7.4 The test examples

The provision of suitable test problems is extremely difficult. Problems that are used to measure the efficiency of algorithms for small dense problems are completely unsatisfactory since the algorithms considered here are intended mainly for large-scale problems. For example, it is pointless to test a truncated-Newton method on a very small problem since the algorithm will be effectively performing a full Newton iteration.

A serious difficulty with using very large test problems is that, for all but the most trivial examples, the CPU time necessary to compute the objective function will be very large. This is typically the case if we attempt to use real-world problems for testing purposes. Moreover, it is desirable that problems be defined in such a way that they may be used by other researchers. Large-scale real-world problems almost invariably are written in a non-portable form or can be run only with vast quantities of numerical data.

In this study we have attempted to compromise on these issues by collecting a set of non-trivial problems that can be run with moderate ease at other installations. Eighteen problems are considered. Of these, 16 problems are of dimension 50 or greater and 7 problems are of dimension 100. It is necessary to present an extensive number of results because the performance of conjugate-gradient-type methods is generally erratic. If we are to identify which strategy gives a true improvement in performance, a wide spectrum of results must be considered.

The test examples may be separated into two classes. The first class contains problems whose Hessian matrix at the solution has clustered eigenvalues; the second contains problems whose Hessian matrix has an arbitrary eigenvalue distribution.

Example 1. Pen1 (Gill, Murray, and Pitfield [1972])

$$F(x) = a\sum_{i=1}^{n}(x_i - 1)^2 + b\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{4}\right)^2.$$

The solution varies with n, but $x_i = x_{i+1}$, $i = 1, \ldots, n-1$. All the runs made were with $a = 1$, $b = 10^{-3}$. With these values, the Hessian matrix at the solution has $n-1$ eigenvalues $O(1)$ and one eigenvalue $O(10^{-3})$. The Hessian matrix is full and consequently, for large values of n, conjugate-gradient type methods are the only techniques available.

Example 2. Pen2 (Gill, Murray, and Pitfield [1972])

$$F(x) = a\sum_{i=2}^{n}\left(\left(e^{x_i/10} + e^{x_{i-1}/10} - c_i\right)^2 + \left(e^{x_i/10} - e^{-1/10}\right)^2\right)$$
$$+ b\left(\left(\sum_{i=1}^{n}(n-i+1)x_i^2 - 1\right)^2 + \left(x_1 - \frac{1}{5}\right)^2\right)^2,$$

where $c_i = e^{i/10} + e^{(i-1)/10}$ for $i = 2, \ldots, n$. The solution varies with n, but $x_i = x_{i+1}$ for $i = 1, \ldots, n-1$. This example was also run with $a = 1$ and $b = 10^{-3}$. For these values the Hessian matrix at the solution has $n-2$ eigenvalues $O(1)$ and two eigenvalues $O(10^{-3})$. The Hessian matrix is full.

Example 3. Pen3 (Gill, Murray, and Pitfield [1972])

$$F(x) = a\Bigg\{1 + e^{x_n}\sum_{i=1}^{n-2}(x_i + 2x_{i+1} + 10x_{i+2} - 1)^2$$
$$+ \left(\sum_{i=1}^{n-2}(x_i + 2x_{i+1} + 10x_{i+2} - 1)^2\right)\left(\sum_{i=1}^{n-2}(2x_i + x_{i+1} - 3)^2\right)$$
$$+ e^{x_{n-1}}\sum_{i=1}^{n-2}(2x_i + x_{i+1} - 3)^2\Bigg\}$$
$$+ \left(\sum_{i=1}^{n}(x_i^2 - n)\right)^2 + \sum_{i=1}^{n/2}(x_i - 1)^2.$$

At the minimum, this function has $n/2$ eigenvalues $O(1)$ and $n/2$ eigenvalues $O(10^{-3})$. The Hessian matrix is full.

The remaining examples have arbitrary distributions of eigenvalues at the solution.

Example 4. Chebyquad (Fletcher [1965])

$$F(x) = \sum_{i=1}^{n} f_i(x)^2,$$

where

$$f_i(x) = \int_0^1 T_i^*(x)dx - \frac{1}{n}\sum_{j=1}^{n} T_i^*(x_j), \qquad i = 1, \ldots, n,$$

and $T_i^*(x)$ is the i^{th}-order shifted Chebyshev polynomial. The Hessian matrix is full.

Example 5. GenRose

This function is a generalization of the well-known two-dimensional Rosenbrock function (Rosenbrock [1960]).

$$F(x) = 1 + \sum_{i=2}^{n} (100(x_i - x_{i-1}^2)^2 + (1 - x_i)^2).$$

Our implementation of this function differs from most others in that $F(x)$ is unity at the solution rather than zero. This modification ensures that the function cannot be computed with unusually high accuracy at the solution and is therefore more typical of practical problems.

The next three examples arise from the discretization of problems in the calculus of variations. Similar problems arise in the numerical solution of optimal control problems. The general continuous problem is to find the minimum of the functional

$$J(x(t)) = \int_0^1 f(t, x(t), x'(t))\, dt,$$

over the set of piecewise differentiable curves with the boundary conditions $x(0) = a$, $x(1) = b$. If $x(t)$ is expressed as a linear sum of functions that span the space of piecewise cubic polynomials then minimization of J becomes a finite-dimensional problem with a tri-diagonal Hessian matrix.

Example 6. Cal1 (Gill and Murray [1973a])

$$J(x(t)) = \int_0^1 \left\{ x(t)^2 + x'(t) \tan^{-1} x'(t) - \log(1 + x'(t)^2)^{\frac{1}{2}} \right\} dt,$$

with the boundary conditions $x(0) = 1$, $x(1) = 2$.

Example 7. Cal2 (Gill and Murray [1973a])

$$J(x(t)) = \int_0^1 \left\{ 100(x(t) - x'(t)^2)^2 + (1 - x'(t))^2 \right\} dt,$$

with the boundary conditions $x(0) = x(1) = 0$.

Example 8. Cal3 (Gill and Murray [1973a])

$$J(x(t)) = \int_0^1 \left\{ e^{-2x(t)^2} (x'(t)^2 - 1) \right\} dt,$$

with the boundary conditions $x(0) = 1$, $x(1) = 0$.

Example 9. QOR (Toint [1978])

$$F(x) = \sum_{i=1}^{50} \alpha_i x_i^2 + \sum_{i=1}^{33} \beta_i \Big(d_i - \sum_{j \in A(i)} x_j + \sum_{j \in B(i)} x_j \Big)^2,$$

where the constants α_i, β_i, d_i and sets $A(i)$ and $B(i)$ are described in Toint's paper. This function is convex with a sparse Hessian matrix.

Example 10. GOR (Toint [1978])

$$F(x) = \sum_{i=1}^{50} c_i(x_i) + \sum_{i=1}^{33} b_i(y_i),$$

where

$$c_i(x_i) = \begin{cases} \alpha_i x_i \log_e(1 + x_i), & x_i \geq 0, \\ -\alpha_i x_i \log_e(1 + x_i), & x_i < 0, \end{cases}$$

$$y_i = d_i - \sum_{j \in A(i)} x_j + \sum_{j \in B(i)} x_j$$

and

$$b_i(y_i) = \begin{cases} \beta_i y_i^2 \log_e(1 + y_i), & y_i \geq 0, \\ \beta_i y_i^2, & y_i < 0. \end{cases}$$

The constants α_i, β_i, d_i and sets $A(i)$ and $B(i)$ are defined as in Example QOR. This function is convex but there are discontinuities in the second derivatives.

Example 11. ChnRose (Toint [1978])

$$F(x) = 1 + \sum_{i=2}^{25} \left(4\alpha_i (x_{i-1} - x_i^2)^2 + (1 - x_i)^2 \right),$$

where the constants α_i are those used in the example QOR. The value of $F(x)$ at the solution has been modified as in Example 5. The Hessian matrix is tri-diagonal.

7.5 Starting points

The starting points used were the following.

Start 1

$$x^{(0)} = (0, 0, \ldots, 0)^T.$$

Start 2

$$x^{(0)} = \left(\frac{1}{n+1}, \frac{2}{n+1}, \ldots, \frac{n}{n+1}\right)^T.$$

Start 3

$$x^{(0)} = (1, -1, 1, -1, \ldots)^T.$$

Start 4

$$x^{(0)} = (-1, -1, \ldots, -1)^T.$$

7.6 Description of the tests

All the algorithms are coded in double-precision Fortran IV. The runs were made on an IBM 370/168, for which the relative machine precision, ϵ, is approximately 10^{-15}.

Each algorithm requires two additional user-specified parameters: λ, the bound upon the change in x at each iteration, (the quantity $\|x^{(k+1)} - x^{(k)}\|_2$) and F_{est}, an estimate of the value of the objective function at the solution. For all problems, the value of λ was set at 10, and F_{est} was set to the value of $F(x)$ at the solution.

For the initial testing, a limited set of test functions was used. This set includes: Pen1 ($N = 50$, Start 3), GenRose ($N = 50$, Start 2), Cal1 ($N = 50$, Start 1), and Chebyquad ($N = 20$, Start 2). These four functions have quite different behavior, and it was found that performance on these test functions was often indicative of the performance of an algorithm on the complete battery of test functions. For these limited tests, only the value $\eta = .25$ was used and an time limit of 15 seconds was placed on each test run.

In order to determine a "good" truncated-Newton algorithm and also to compare the performance of this good truncated-Newton algorithm against better-known algorithms, more complete tests were carried out. The complete set of test functions was used, and the values $\eta = .25, .1, .001$ were tried. Many of these numerical results were obtained by Gill and Murray [1979]. Since the optimal value of η is often larger for algorithms which use second-derivative information, a series of tests with $\eta = .5, .7, .9$. was also carried out. Finally, a special set of comparisons against Newton's method was done.

The full set of results is contained in the tables in the appendix. Each entry in a

table consists of a pair of values: the first is the number of non-linear iterations required to find the solution, the second is the number of function/gradient evaluations (unless otherwise indicated, this number includes the function evaluations in the linesearch as well as those used to compute the matrix-vector products in the linear sub-algorithm). A number of the test functions have sparse Hessian matrices, and in these cases it is possible to use sparse finite-differencing to compute these matrices at the beginning of each non-linear iteration. A lower-case "s" at the end of a function name (for example, GenRs) indicates that sparse finite-differencing is being used. (The routines for computing the sparse Hessian matrices were developed by Thapa [1980].)

7.7 Discussion of results

The first tests were used to determine the better preconditioning strategies. These are summarized in table 1. The routine used was a preconditioned Lanczos algorithm with forcing function (4.3.6) and the standard forcing sequence (4.4.3). PLMA was the non-linear outer algorithm. The terms used to describe the preconditioning strategies correspond to section 8.2.3; the letters DNC indicate that the algorithm did not converge in 15 CPU seconds.

The results indicate that the diagonal preconditionings are the most effective. The exact diagonal of the Hessian often performs very well; the only exception is for the function GenR, where the Hessian matrix is frequently indefinite. A better strategy for handling negative diagonal elements might improve the result in this case (for these runs, negative diagonal elements were replaced by their absolute value).

Tables 2 and 3 show the effects of different forcing sequences. The exact BFGS diagonal preconditioning was used in combination with the routine described for table 1. Table 2 was made using the forcing function (4.3.6) and table 3 with function (4.3.5). The tests in table 2 follow naturally from the discussion in section 4.4; those in table 3 were made because the standard forcing sequence ($\phi_k = \min\{1/k, \|g^{(k)}\|\}$) was found to be too stringent in combination with function (4.3.5).

Our experience with alternative forcing sequences has been inconclusive; the tests in Table 2 were included to show how well they sometimes performed on specific functions. The standard forcing sequence is quite effective for the class of problems chosen. Table 3 shows that scaling this forcing sequence by 1.5 is worthwhile when function (4.3.5) is

being used.

Tables 4A–4E were used to choose the optimal truncated-Newton routine. The three approximate diagonal preconditionings were used on all test problems with $\eta = .25$ in combination with the following routines:

1. TN1—a preconditioned Lanczos algorithm with the standard forcing sequence and forcing function (4.2.2),
2. TN2—a preconditioned Lanczos algorithm with the standard forcing sequence scaled by 1.5 and forcing function (4.3.5),
3. TN3—as in TN1, but with forcing function (4.3.6),
4. MINR—a preconditioned MINRES algorithm.

All routines use PLMA as the non-linear outer algorithm. The numbering of the preconditionings corresponds to the list in section 8.2.3.

The totals from all the runs are listed in table 4E. The best routine could be decided upon in a number of ways:

1. iteration count
2. function evaluations (regular)
3. function evaluations (sparse)
4. function evaluations (total)
5. function evaluations (regular) plus iteration count
6. function evaluations (sparse) plus iteration count
7. function evaluations (total) plus iteration count

With the exception of criteria 2 and 5 where TN2 PC=1 is best, the totals indicate that TN1 PC=3 is the optimal routine.

In order to ascertain the overall effectiveness of truncated-Newton algorithms, routine TN1 PC=3 (hereafter referred to simply as TN) was compared with PLMA, MNM, and QNM on the full set of test functions for $\eta = .25, .1, .001$. In addition, a very simple truncated-Newton algorithm was examined both with and without an exact BFGS diagonal preconditioning (see section 8.3.2). These routines are referred to as PBTN and BTN, respectively (the initials P and B stand for "preconditioned" and "basic"). The results of these tests can be found in tables 5A–5E; NR indicates that a test was not run, and NA that a total is not available.

As table 5E indicates, TN only requires 50% to 80% as many function evaluations

as PLMA to solve the full set of test problems. This seems a significant reduction. When sparse finite-differencing is used, both PBTN and BTN can compare favorably with TN. Without this feature, however, they are considerably slower. This is especially true of the unpreconditioned routine BTN; a major factor is the performance of this routine on the problem Cal1 $N = 100$.

The set of tests summarized in tables 6A and 6B shows the performance of routine TN with different values of η. Regardless of which performance measure is used, .25 is always the overall optimal value of η for this routine. The results were insensitive to the various choices of η. This is in marked contrast to PLMA. The main reason for this insensitivity was that the initial step was close to the minimum along the search direction.

The final set of tests, summarized in tables 7A and 7B, are a special comparison of the truncated-Newton and modified-Newton algorithms. When TN and MNA were compared in the tables 5, the work required to compute second-derivative information was ignored for MNA but counted for TN, even though TN only computes partial second-derivative information whereas MNA computes the full Hessian matrix. In the tables 7, these two routines are evaluated in a fairer way. It is assumed that, at the beginning of each nonlinear iteration, the full Hessian is evaluated and then used either to solve the Newton equations (in the case of MNA) or to compute the necessary matrix/vector products (for TN). As a result, in the number pairs in the tables, the first number indicates the number of Hessian matrices computed, and the second the number of function/gradient evaluations used in the linear search. As the totals indicate, the truncated-Newton algorithm requires fewer Hessian matrices as well as fewer function/gradient evaluations. In fact, TN is almost twice as efficient as MNA. This is especially surprising since TN is a routine designed for large-scale function minimization and not for general optimization, like MNA.

7.8 A supplementary test problem

Up until this point, all of the algorithms have been tested on a particular set of test problems. This raises the question of whether a good truncated-Newton algorithm has been found or whether we have just found the optimal algorithm for this set of test functions. In addition, for practical reasons we have limited ourselves to relatively

small test functions ($n \leq 100$). For this reason, we now test algorithm TN on a larger, independent test example.

The function is taken from Murtagh and Saunders [1980]. It investigates the optimal control of a spring, mass, and damper system. In its original form, the problem has a quadratic objective function and a set of equality and inequality constraints:

$$\min f(x,y,u) = \frac{1}{2}\sum_{t=0}^{T} x_t^2$$

subject to

$$\begin{aligned} x_{t+1} &= x_t + 0.2y_t \\ y_{t+1} &= y_t - 0.01y_t^2 - 0.004x_t + 0.2u_t \\ -0.2 &\leq u_t \leq 0.2 \\ y_t &\geq -1.0 \end{aligned}$$

for $t = 0, \ldots, T-1$, and

$$x_0 = 10, \qquad y_0 = 0, \qquad y_T = 0.$$

The starting point used was $x_t = 0$, $y_t = -1$ $(t = 0, \ldots, T)$, and $u_t = 0$ $(t = 0, \ldots, T-1)$. For these tests, $T = 100$, and so there are 302 variables in all.

Since this is a constrained optimization problem, and algorithm TN is only designed to solve unconstrained problems, we minimize a related penalty function:

$$F(x,y,u) = \rho f(x,y,u) + c^T c.$$

Here, c is a vector with one component for each constraint above. For example, $c_1 = x_1 - x_0 - 0.2y_0$. If c_i is a component corresponding to an inequality constraint such as $y_t \geq -1.0$, then $c_i = y_t + 1.0$ if $y_t < -1.0$, and $c_i = 0.0$ otherwise. The parameter ρ is a penalty coefficient; the smaller its value, the more stringently the constraints must be satisfied. For our tests, ρ was set equal to 10^{-3}, 10^{-5}, and 10^{-7}. The minimal value of the objective function f subject to the given constraints is 1186.382.

Setting the penalty parameter $\rho = 10^{-3}$ was not sufficient for our purposes, since the minimum of the penalty function was quite different from the minimum of the original function; in this case, the final value of f was 729.2, not even close to the optimal value. There were also problems with $\rho = 10^{-7}$, but for quite different reasons. Recall that the convergence criterion for the algorithm is given by (7.2.1), where $\tau = 10^{-5}$. Here, $\rho \cdot f_{min} = \rho \times 1186.382 = 1.2 \times 10^{-4}$, so that only two digits of the optimal function

value were obtained.

For $\rho = 10^{-5}$, the final computed function value was $f = 1190.384$, which is close to the optimal value of the constrained function. The value of c^Tc, the square of the norm of the constraint violations, was approximately 10^{-4}. When second derivatives were available to compute the matrix/vector products, algorithm TN required 168 function/gradient evaluations to minimize this penalty function. Murtagh and Saunders [1980], using a projected Lagrangian algorithm, required 203 function/gradient evaluations to obtain a solution with $c^Tc \leq 10^{-12}$. Although these two results are not directly comparable, they do indicate that the truncated-Newton algorithm is effective in solving this problem. When the matrix/vector products were computed using finite differencing, algorithm TN required 1727 function/gradient evaluations to minimize F. If sparse finite-differencing had been used to approximate the Hessian, 423 function/gradient evaluations would have been used (each Hessian can be computed using five gradients).

There is reason to assume that truncated-Newton algorithms will in general perform well on penalty functions. Because of the special form of F, the Hessian matrix will often have two clusters of eigenvalues. The first, corresponding to the objective function $\rho \cdot f$, will be $O(\rho)$; the second, corresponding to the penalty term, will be $O(1)$. The Lanczos algorithm, applied to the soluton of the Newton equations, works well if the matrix has only a few clusters of eigenvalues. Also, the Lanczos algorithm is able to quickly and accurately approximate the extreme eigenvalues of a matrix (see Parlett [1980], section 12-5). Hence, if a truncated-Newton algorithm is applied to a penalty function, where at each stage the Newton equations involve a matrix whose eigenvalues fall into two clusters at the ends of the spectrum, good performance should result.

8 Adapting Truncated-Newton Methods

8.1. Introducton

When Truncated-Newton methods were presented in Chapter 3, the basic algorithm was deliberately left vague. This was because there are many ways in which such an algorithm can be implemented. At each step, a choice must be made about how a certain result or effect is to be achieved.

Some possible choices were outlined, or at least mentioned, in the succeeding chapters. In Chapter 3, algorithms for approximately solving the Newton equations were developed. In Chapter 4, we described ways of terminating the linear algorithm. And in Chapter 5, it was shown how the method could be preconditioned using other linear and non-linear methods.

When designing a program for a specific computer, or when choosing a method to solve a specific problem, decisions must be made about which method to use and how it will be implemented. In the case of a truncated-Newton method, many rather detailed options have to be selected in order to obtain a usable algorithm.

Often, the first question asked is which algorithm is the most efficient for solving the given problem or a wide class of problems. Answers to this question are usually based on numerical tests, which were the subject of Chapter 7. But this is not the only criterion for selecting an algorithm. Another important question is which method is most stable. This question can sometimes be answered absolutely on the basis of theoretical results from perturbation theory.

Many other questions arise because of more practical issues such as: (a) the expense of computing the function being minimized, (b) the availability of second derivatives, (c) the size of the computer, (d) the availability of routines in a program library, (e) the number of times a problem is to be solved, etc.

Until recently, it was generally assumed that researchers would be working on a large central computer, and that professionally written software would be available on-line in a subroutine library. With the rise of the small-computer industry, this assumption is now often false, and it is now necessary to take into account the effect of small machines when designing algorithms. On a small computer, the size of the program can be as important a consideration as the size of the problem. This is not just because storage

space is at a premium: numerical program libraries for small machines are still rare, and the user must often write his own programs or manually input commercial programs. Short and simple algorithms can greatly reduce the likelihood of error.

In addition, small computers are often owned by the user, and are generally used by a small group of people only. This means that a routine considered slow in a large-machine environment can be attractive if it offers a substantial reduction in storage requirements. It could be left to run for long periods, for example overnight, with little inconvenience. This can greatly influence the choice of an algorithm; the optimal routine for a large machine can have little resemblance to the optimal routine for a mini-computer.

We mentioned above several questions related to the actual problem being solved—the difficulty of computing the function, and the availability of second-derivative information, for example. Choosing a method based on these criteria often depends on the efficiency of the method, and the choice must be made on the basis of numerical tests. Some decisions, however, can be made *a priori*, such as general decisions about solving the Newton equations and about how matrix/vector products are to be computed.

In order to simplify the process of choosing a specific algorithm, we summarize in section 8.2 the possibilities for a truncated-Newton algorithm. There we list the choices for each step of the algorithm, indicate operation and storage counts, describe possible interactions with other modules in the method, and mention difficulties that might be encountered in programming. In 8.3, a couple of sample situations are described, along with suggestions about how to put together a truncated-Newton algorithm which is well-suited to the needs of each case.

8.2. Choices for sub-algorithms

In order to specify a truncated-Newton algorithm, five sub-algorithms must be selected. These are:

1. The algorithm to approximately solve the Newton equations (2.2.3).
2. The non-linear outer algorithm.
3. The linear preconditioning strategy.
4. The termination criterion and forcing sequence for the linear algorithm.
5. The algorithm for computing the Hessian/vector products.

These sub-algorithms have been discussed at length in preceding chapters, but mainly

from a theoretical point of view. In this section, we shall take up issues which would arise when programming and using truncated-Newton methods.

In the succeeding sub-sections, we will examine each of these choices separately, indicating operation and storage counts. The notation for vectors is global, so that if a vector name appears in two sections, the same vector is being referred to and no additional storage is required. Generally, all choices may be made independently, but usually only one choice may be made from each section. Exceptions to this rule will be noted as they occur.

For reference, here is a list of the vectors used below:

x	the current estimate of the minimizing point F
p	search direction
g	gradient
p_0	initial search direction, used in Beale's method
D	vector which represents diagonal preconditioning matrix
D_1	temporary value of D
E	represents sub-diagonal of preconditioning matrix
E_1	temporary value of E
s_i	scratch vectors

8.2.1. Approximately solving the Newton equations

We shall consider six ways of approximately solving the Newton equations (2.2.3):

1. conjugate-gradient (section 2.4)
2. preconditioned conjugate-gradient (Concus et. al [1976])
3. Lanczos (sections 3.3–3.5)
4. preconditioned Lanczos (sections 3.3–3.6, 3.8)
5. MINRES (section 3.7)
6. preconditioned MINRES (sections 3.7–3.8)

The storage requirements and operation counts for these methods are summarized in Table 8.1 below. Some problems that might be encountered when using these methods are also mentioned briefly there. For the reasons given in section 3.2, SOR-related methods will not be examined. Clearly, only the preconditioned algorithms can be used in combination with a non-linear algorithm or a linear preconditioning scheme.

The conjugate-gradient algorithm is the simplest algorithm that we consider feasible for solving the Newton equations. Unfortunately, the conjugate-gradient method is only designed to solve systems of equations with positive-definite matrices. In an optimization setting where the Hessian matrix in the Newton equations can be indefinite, this is a serious deficiency; but if a given problem is known to have a positive-definite Hessian everywhere, this may not matter. When the Hessian is indefinite, the conjugate-gradient method may be unstable. The addition of preconditioning can greatly improve the performance of this method at little computational cost. Therefore, except under extreme circumstances, a preconditioned conjugate-gradient method is always to be preferred over the regular conjugate-gradient method.

ALGORITHM	STORAGE	OPERATIONS	COMMENTS
conjugate-gradient (cg)	p, $s_1 - s_3$	$8n$, matrix-vector product	only for positive-definite systems
preconditioned cg	p, $s_1 - s_4$	$8n$, matrix-vector product, preconditioning step	only for positive-definite systems
Lanczos	p, $s_1 - s_4$	$12n$, matrix-vector product	complex to program
preconditioned Lanczos	p, $s_1 - s_5$	$12n$, matrix-vector product, preconditioning step	complex to program
MINRES	p, $s_1 - s_5$	$16n$, matrix-vector product	complex to program
preconditioned MINRES	p, $s_1 - s_6$	$16n$, matrix-vector product, preconditioning step	complex to program

Table 8.1 Choices for the linear algorithm

In order to be able to treat indefinite systems of equations, it is possible to use a method based on the Lanczos algorithm for tridiagonalizing a symmetric matrix. It is slightly more expensive to use than the conjugate-gradient method; it is also more complex to program since it involves three separate sub-algorithms: the Lanczos tridiagonalization, the modified-Cholesky factorization, and the conjugate-gradient step.

In a general setting, though, it is a stable and predictable way of handling indefinite Hessian matrices, which is not true of the conjugate-gradient algorithm. Again, it is easy to add a preconditioning step.

The MINRES algorithm is a variant of the Lanczos algorithm which guarantees that the norm of the residual decreases at each iteration. It is based on a QR factorization of the tridiagonal matrix resulting from the Lanczos process. Programming this method is comparable to programming the Lanczos method above, and it is equally easy to precondition the algorithm.

8.2.2. Non-linear algorithms

We shall look at five non-linear outer algorithms:

1. linesearch (section 1.4)
2. non-linear conjugate-gradient (section 2.4)
3. Beale's method (Gill and Murray [1979])
4. limited-memory quasi-Newton (section 2.5, Gill and Murray [1979])
5. quasi-Newton (section 2.3)

Describing a linesearch as a non-linear outer algorithm may be something of an overstatement. We are referring to the following method: 1) approximately solve the Newton equations at the current point to compute a search direction, 2) use this search direction in the linesearch to compute a new point.

ALGORITHM	STORAGE	OPERATIONS	COMMENTS
linesearch	p, g, x, s_1	$\geq 3n$,	difficult to program
non-linear cg	p, g	$3n - 4n$,	requires a linesearch
Beale's method	p, p_0, g	$8n - 10n$,	requires a linesearch
limited-memory quasi-Newton method (k updates)	p, g, $s_1 - s_{2k}$	$k(k+1)/2 \times$ (one update)	requires a linesearch, may be preconditioned

Table 8.2 Choices for the non-linear algorithm

With the exception of the line-search, all of these algorithms are used to generate a preconditioning for the linear algorithm, i.e. the formulas for the outer algorithm

implicitly describe some linear operator which can then be applied to any vector. Thus, these algorithms can be used only with an algorithm which can be preconditioned. Trust-region methods could also be used as non-linear algorithms for a truncated-Newton code, but they will not be discussed here. A summary of operation counts and storage requirements can be found in Table 8.2 above.

A linesearch is the simplest algorithm that is quaranteed to converge that could be used for the non-linear outer iteration in a truncated-Newton code; also, a linesearch will be a part of all the other algorithms to be described in this sub-section. Thus, such an algorithm would be central to any program using a linesearch stategy. The operation count is difficult to estimate, since it will depend on how many guesses are needed to "sufficiently decrease" the value of the objective function. If k guesses are used, then $(2k + 1)n$ operations and k function-gradient evaluations will be required. For many problems, k will be equal to 1 as the minimum is approached. Our numerical tests have indicated that truncated-Newton methods compute well-scaled search directions, and that k is often equal to 1 when the Hessian matrix is positive-definite. An efficient linear search can be difficult to program, but sample programs are often found in program libraries and even on pocket calculators.

The simplest way to generate a preconditioning for the linear algorithm is to use a non-linear conjugate-gradient algorithm. Beale's method is a variant of a non-linear conjugate-gradient algorithm in which the new search direction is computed using both the most recent direction as well as the first direction.

Limited-memory quasi-Newton algorithms are almost as flexible as truncated-Newton methods, because it is possible to choose both the type of quasi-Newton update to use as well as the total number of updates. There is some evidence to indicate (see Fenelon [1981]) that choosing k bigger than 2 is not economical. It should be noted that a diagonal preconditioning can be added to this algorithm (section 5.3).

It is also possible to consider using a quasi-Newton method to precondition the linear algorithm. Unlike all the other methods considered in this section, quasi-Newton methods have storage and operation counts which are quadratic, not linear, in n. For this reason, it is difficult to imagine them being competitive with the other methods proposed here.

8.2.3. Linear preconditionings

Most of the preconditionings generated during the linear subiteration were discussed in detail in Chapter 5. However, a few of them were only alluded to in passing. In this sub-section we will consider the following options:

1. BFGS diagonal preconditioning (section 5.3)
2. rank-one diagonal preconditioning (section 5.3)
3. exact BFGS diagonal preconditioning (sections 5.3 and 5.4)
4. exact diagonal of the Hessian
5. tridiagonal preconditioning based on VTV^T (section 5.5)
6. tridiagonal preconditioning based on $R^{-T}TR^{-1}$ (section 5.5)
7. product of the tridiagonal preconditionings (section 5.6)
8. exact BFGS tridiagonal preconditioning
9. exact BFGS tridiagonal factors preconditioning

As in the previous section, these options can only be used with a preconditioned linear algorithm. Their operation counts and storage requirements are summarized in Table 8.3 below.

Because a new preconditioning is being developed while the old one is still in use, two copies of the operator must be kept when using all but the fourth preconditioning algorithm. Since the rank-one formula does not guarantee positive-definiteness for the preconditioning, some strategy must be designed to modify the diagonal when negative elements arise. The exact BFGS formula requires a separate initialization step if a non-linear preconditioning is also being used (see section 5.3).

The two tridiagonal preconditionings (options 5 and 6 above) are very similar. They have the same storage requirements, and the programs which implement them have only minor differences. When using option 6, however, it is easy to re-orthogonalize the new Lanczos vector using the projection matrix R. Loss of orthogonality can seriously degrade the performance of conjugate-gradient and Lanczos algorithms; re-orthogonalization can significantly improve stability and convergence.

It is only possible to precondition using the diagonal elements of the Hessian if these elements can be computed at little cost. If this is feasible, then this is a simple and inexpensive preconditioning to use. If the Hessian is not positive definite everywhere, some strategy must be devised to handle negative diagonal elements. They might be set

to some small positive value, or their absolute value might be used; it is also possible to use the negative diagonals to compute directions of negative curvature. This latter option could be used in place of the inner algorithm at the current iteration. A direction of negative curvature could be computed immediately from the exact Hessian information and used in the line search with little cost.

ALGORITHM	STORAGE	OPERATIONS	COMMENTS
BFGS diagonal	D, D_1, s_1	$9n$ to update, n to apply	
rank-one diagonal	D, D_1, s_1	$6n$ to update, n to apply	may be indefinite
exact BFGS diagonal	D, D_1	$8n$ to update, n to apply	requires initialization
diagonal of Hessian	D	n to apply	may be costly to obtain, may be indefinite
tridiagonal VTV^T (k projectors)	$s_1 - s_{2k+1}$	$(4k+1)n$ to apply, $(k^2+k)n$ to form	requires long program
tridiagonal $R^{-T}TR^{-1}$ (k projectors)	$s_1 - s_{2k+1}$	$(4k+1)n$ to apply, $(k^2+k)n$ to form	requires long program
tridiagonal product (k projectors)	$s_1 - s_{2k+1}$	$(4k+1)n$ to apply, $(k^2+k)n$ to form	requires long program
BFGS exact tridiagonal	D, D_1, E, E_1	$6n$ to apply, $14n$ to update	may be indefinite

Table 8.3 Possible preconditioning strategies

The next three preconditionings are considerably more complex and expensive to apply than any of the other preconditioning strategies examined here. This is because they involve the projected Hessians, and information about the projection matrices must be computed and stored. There is also some choice about how much information will be used; we assume here that k Lanczos vectors are needed, and that k is small in comparison with n. Since these preconditionings use information from the Lanczos algorithm, they cannot be used with the regular linear conjugate-gradient method.

It is possible to use the correspondence between the quasi-Newton and linear conjugate-gradient algorithms to generate not just the diagonal, but also the principle

subdiagonal of Hessian matrix. This method is almost as easy to program as the diagonal preconditionings above; however, there is a problem with positive-definiteness. Although the diagonal and the complete Hessian approximation can be guaranteed to be positive definite, the tridiagonal submatrix may be indefinite, and some strategy must be derived for modifying it in this case.

Because of the problem with indefiniteness for the preceding method, it would be preferable to update the diagonal and subdiagonal of the Cholesky factor of the approximate Hessian. This would guarantee a positive-definite preconditioning. Unfortunately, this is infeasible. An examination of the formulas for updating matrix factorizations in Gill, et al. [1974] shows that updating a portion of the factorization other than the diagonal requires knowledge of the complete factorization of the old preconditioning. Even if this information were available, accessing it would be an $O(n^2)$ process. Since the preconditioning is being updated at every linear sub-iteration, this method is uneconomical. Because generally very few linear iterations are performed, even making an update of this type once per outer iteration would often be impractical.

8.2.4. Termination criteria for the linear algorithm

In Chapter 4, we considered the following convergence criteria for the linear algorithm:

1. $\|r_q\|/\|g^{(k)}\|$
2. $|Q_{q+1} - Q_q|^{\frac{1}{2}}/|Q_1|^{\frac{1}{2}}$
3. $|Q_{q+1} - Q_q|^{\frac{1}{2}}/\|g^{(k)}\|$
4. $|Q_{q+1} - Q_q|^{\frac{1}{2}}/|Q_{q+1}|^{\frac{1}{2}}$

All of these formulas require $2n$–$3n$ operations to compute. In some cases, for example criterion 1 in combination with a MINRES algorithm, they are a natural by-product of the algorithm. When the Lanczos algorithm is used for the linear sub-iteration it may be necessary to compute and store the residual in order to apply these tests. They are all easy to program. The choice of a forcing sequence (see section 4.4) can be made solely on the basis of numerical tests.

8.2.5. Computing matrix/vector products

There are three principle methods of obtaining the matrix/vector products Gp required during the linear sub-iteration. They are:

1. finite differencing along p
2. computing G using (sparse) finite-differencing (section 2.5)
3. computing G

Which method is used depends on the function being minimized; it does, however, have an important bearing on the remainder of the algorithm. If the Hessian is difficult to compute, or is large and dense, then finite differencing along p may be the only option available. This discourages the use of a large number of linear-subiterations since an additional gradient evaluation is required for each matrix/vector product. For problems where the function and its first and second derivatives are inexpensive to compute relative to the cost of solving the linear system, we would again perform few linear iterations, as the cost of the linear sub-algorithm would dominate the cost of the function and gradient evaluations.

When G is available and the function is moderately expensive to compute, a larger number of inner iterations would be encouraged. In this case, the cost of computing the function and its derivatives dominates the cost of the linear sub-iteration and is the same at every non-linear outer iteration.

Unfortunately, few absolute statements can be made about choosing this segment of the truncated-Newton algorithm; a decision should be made in the context of a specific problem or class of problems.

8.3. Choosing a complete truncated-Newton algorithm

In this section, we will describe what we consider to be the two extreme versions of a truncated-Newton algorithm. The decisions made about the construction of the complete algorithm are based mainly on the size of the machine being used—either very small or large. The ideas used to describe each of these situations can be easily applied to more specialized cases.

In the introduction to this chapter, we mentioned a number of issues which might affect the choice of a particular algorithm. Some of these involved the function being

minimized. Although a truncated-Newton algorithm can be used for general optimization, we consider that it will be most useful for large-scale minimization problems.

In this context, we take "large-scale" to mean that it is difficult to use second-derivative information. This might be because the dimension of the problem is large, in which case storing or factoring the second-derivative matrix is impossible. It might also mean that the Hessian matrix is expensive to compute or is unavailable. A further possibility is that the Hessian matrix may be expressed as the product of several large sparse matrices, and it is uneconomical to obtain its elements explicitly (this is the case in large constrained optimization problems (see Chapter 6)).

8.3.1 The large-machine case

When working on a large machine, the only important consideration is efficiently finding the solution to the problem in a stable manner. All the necessary algorithms are assumed to be professionally coded and available in a program library, and the size of the program is not an issue (since it is pre-compiled in an object-code library). The length and complexity of the algorithms are not factors in any decision. However, the storage requirements for the method (the number of vectors required) are still important.

Thus, a preconditioned Lanczos algorithm should be selected to approximately solve the Newton equations. Alternatively, a preconditioned MINRES algorithm might be chosen if it could be shown to be more effective for the class of problems being solved; when terminated using $\|r_q\|$, MINRES is almost as efficient as a Lanczos method. As a non-linear algorithm, we would probably select a two-step diagonally-preconditioned limited-memory quasi-Newton method. Such an algorithm has been shown to be efficient and cost-effective for large optimization problems (see Gill and Murray [1979], Fenelon [1981]), and has performed well in our numerical tests here.

We would choose one of the diagonal preconditioning schemes to precondition the linear algorithm. They all have low storage requirements and are inexpensive to generate. The tridiagonal preconditioning schemes are considerably more expensive to use and less successful in practise; they would have to perform much better in numerical tests before they could be recommended for general use. Our results indicate that the exact BFGS diagonal preconditioning is the most effective of the diagonal schemes. This choice is based on numerical tests, storage requirements, and the stronger theoretical justifications

for this scheme. Since we are unconcerned about the length of the program, it would be possible to include preconditioning with the diagonal elements of the Hessian as a user-specified option when these elements can be computed easily.

The other options for the algorithm (the termination criterion and the forcing sequence) would be chosen on the basis of numerical tests. It would depend somewhat on the other choices made for the algorithm. The method used to compute matrix/vector products could be chosen by the user of the code at run-time.

8.3.2 The small-machine case

The major difference between the small- and the large-machine cases is that the size of the program is now a factor in the choice of the algorithm. It is impossible to store a large code in the memory of a small machine. For this reason, simple iterative methods are often preferable to direct methods for solving many problems. Also, because the user is often not paying for computer time, and time-sharing is not in effect, storage can be a more important issue than speed in the choice of an algorithm.

Another reason to favor simple and short algorithms is that a small computer will not usually come complete with a program library. The user must either write his own programs, or at least may be obliged to input the program by typing. In order to decrease the probability of error, and also to reduce the overall time needed to solve a problem, easy-to-program methods are preferred.

For these reasons, a preconditioned conjugate-gradient algorithm could be chosen to approximately solve the Newton equations. In extreme cases, the preconditioning step could be omitted; it is, however, a simple addition to the program and it can greatly speed convergence. A linesearch could be chosen as the non-linear algorithm. Because of possible indefiniteness, the search direction p should be monitored at every linear iteration to insure that it is a descent direction. Although a simple non-linear conjugate-gradient method is easy to add, effective methods of this type include such features as restarting strategies which can increase the complexity of the code.

As in the large-machine case, a diagonal preconditioning should be used, unless storage is at such a premium that no preconditioning can be included. The limitations of the small machine do not seriously affect the choice of the termination criterion and the forcing sequence. The matrix/vector products would probably be computed by finite

differencing along p, for reasons of simplicity.

The analysis of these two special cases gives some indication of how a truncated-Newton algorithm can be adapted to a specific computing environment. Final decisions about preconditioning and termination rules must be made on the basis of numerical tests. Some recommendations were made on the basis of the results in Chapter 7. When solving specific classes problems in special environments, though, some of the detailed choices might be made differently.

Appendix

The following tables summarize the results of the tests discussed in section 7.7. Chapters 7 and 8 may be useful in interpreting the tables given here.

Table 1—Comparison of preconditioning strategies using a subset of the test functions.

Preconditioning	Pen1		GenRs		GenR		Calls		Call		Cheb	
Rank-2 D	7	32	33	183	33	315	12	94	12	191	8	83
Rank-1 D	7	32	34	192	34	338	13	101	13	236	8	84
BFGS D	7	32	34	188	34	316	12	93	12	194	9	91
Exact D	5	24	39	224	39	355	11	78	11	63	8	77
VTV^T	DNC		DNC		DNC		DNC		DNC		17	232
$R^{-T}TR^{-1}$	8	40	DNC		DNC		DNC		DNC		21	230
Product T	10	57	DNC		DNC		DNC		DNC		DNC	
BFGS T	10	174	41	223	41	321	12	93	12	181	14	181

Table 2—Comparison of forcing sequences $\{\psi_k\}$ using a subset of the test functions. $\phi_k = \min\{1/k, \|g^{(k)}\|\}$, and the forcing function (4.3.6) is being used.

Forcing Sequence	Pen1		GenRs		GenR		Calls		Call		Cheb	
$\psi_k = .5$	8	42	39	204	39	247	29	213	29	164	9	56
$\psi_k = .1$	8	44	32	175	32	297	8	59	8	164	7	83
$\psi_k = .05$	8	44	35	207	35	427	7	51	7	172	6	79
$\psi_k = .01$	8	45	33	197	33	149	5	38	5	140	10	152
$\psi_k = \max\{\phi_k, .5\}$	7	31	41	223	41	275	31	228	31	169	9	56
$\psi_k = \max\{\phi_k, .1\}$	7	32	34	188	34	301	13	99	13	180	9	79
$\psi_k = \max\{\phi_k, .05\}$	7	32	35	191	35	317	12	93	12	173	9	90
$\psi_k = \max\{\phi_k, .01\}$	7	32	34	188	34	316	12	93	12	194	9	101
MAXIT=5	7	32	35	181	35	235	DNC		DNC		11	69
MAXIT=10	7	32	35	194	35	316	19	141	19	189	9	79
MAXIT=15	7	32	34	188	34	316	15	113	15	187	9	92
MAXIT=20	7	32	34	188	34	316	14	106	14	201	9	91

Table 3—Comparison of forcing sequences $\{\psi_k\}$ using a subset of the test functions. $\phi_k = \min\{1/k, \|g^{(k)}\|\}$, and the forcing function (4.3.5) is being used.

Forcing Sequence	Pen1		GenRs		GenR		Cal1s		Cal1		Cheb	
$\psi_k = 1.0\phi_k$	7	32	37	207	37	391	10	77	10	177	11	130
$\psi_k = 1.5\phi_k$	7	32	35	183	35	342	10	77	10	163	8	80
$\psi_k = 2.0\phi_k$	7	32	34	190	34	344	11	88	11	181	10	96
$\psi_k = 2.5\phi_k$	7	32	38	210	38	402	11	84	11	168	8	59
$\psi_k = 3.0\phi_k$	7	32	38	205	38	347	13	102	13	164	10	90

Tables 4—Comparison of a number of truncated-Newton routines with various diagonal preconditionings. The full set of test functions is used with $\eta = .25$.

Table 4A—Smaller functions. Differencing along search direction.

Function	PC	TN1		TN2		TN3		MINR	
Pen1	1	7	29	7	32	7	32	7	38
Start 3	2	7	29	7	32	7	32	7	38
$n = 50$	3	7	29	7	32	7	32	7	39
Pen2	1	11	55	10	56	10	57	16	93
Start 3	2	10	49	10	56	10	56	17	100
$n = 50$	3	11	64	10	58	10	59	17	102
Pen3	1	10	49	10	47	9	39	10	53
Start 3	2	10	54	10	48	9	45	11	67
$n = 50$	3	10	47	10	47	10	45	11	66
GenR	1	31	370	34	383	33	315	37	245
Start 2	2	32	417	36	519	34	338	38	435
$n = 50$	3	31	330	35	342	34	316	36	329
Cal1	1	9	176	10	175	12	191	11	182
Start 1	2	10	273	10	191	13	236	11	235
$n = 50$	3	10	199	10	163	12	194	12	200
Cal2	1	7	70	6	58	9	69	11	103
Start 1	2	8	87	8	79	10	84	11	141
$n = 50$	3	7	69	7	66	10	86	9	80
Cal3	1	7	112	9	99	10	122	8	97
Start 1	2	7	107	11	96	11	127	9	124
$n = 50$	3	7	112	11	101	11	114	9	123

Table 4B—Larger functions. Differencing along search direction.

Function	PC	TN1		TN2		TN3		MINR	
Pen1	1	2	11	2	11	2	11	2	12
Start 3	2	2	11	2	11	2	11	2	12
$n = 100$	3	2	11	2	11	2	11	2	12
Pen2	1	6	33	5	24	5	27	6	36
Start 3	2	6	32	5	24	5	27	6	36
$n = 100$	3	6	29	5	25	5	26	6	42
Pen3	1	11	68	10	61	10	59	11	69
Start 3	2	11	115	11	99	10	59	11	64
$n = 100$	3	11	65	10	58	10	64	11	62
GenR	1	61	764	63	754	61	683	65	610
Start 2	2	63	1090	64	1046	68	979	65	653
$n = 100$	3	57	684	62	796	64	672	67	1183
Cal1	1	11	413	10	302	14	335	12	351
Start 1	2	10	511	12	521	18	483	14	464
$n = 100$	3	10	409	11	332	15	375	16	1228
Cal2	1	7	154	7	115	11	141	12	166
Start 1	2	9	219	7	142	13	140	15	318
$n = 100$	3	8	117	6	107	11	131	10	407
Cal3	1	7	154	13	194	13	199	10	237
Start 1	2	7	171	11	220	14	232	11	259
$n = 100$	3	7	159	16	229	14	207	11	498

Table 4C—Miscellaneous smaller functions. Differencing along search directions.

Function	PC	TN1		TN2		TN3		MINR	
Cheb	1	8	74	8	83	8	83	10	79
Start 2	2	8	72	8	89	8	84	10	94
$n = 20$	3	7	53	8	80	9	91	10	81
QOR	1	6	25	6	25	6	24	7	32
Start 1	2	6	25	6	24	6	25	7	34
$n = 50$	3	6	25	6	24	6	24	7	38
GOR	1	7	56	8	65	8	60	9	77
Start 1	2	7	59	8	87	8	68	9	80
$n = 50$	3	7	60	8	65	8	57	9	111
ChaR	1	14	96	14	96	15	106	16	117
Start 4	2	14	156	15	133	15	154	15	122
$n = 25$	3	11	77	12	77	12	77	16	151

Table 4D—Sparse finite-differencing.

Function	PC	TN1		TN2		TN3		MINR	
GenRs	1	31	179	34	193	33	183	37	187
Start 2	2	32	190	36	204	34	192	38	195
$n = 50$	3	31	168	35	183	34	188	36	184
GenRs	1	61	357	63	356	61	348	65	339
Start 2	2	63	367	64	376	68	409	65	326
$n = 100$	3	57	335	62	348	64	379	67	356
Cal1s	1	9	71	10	77	12	94	11	87
Start 1	2	10	78	10	77	13	101	11	89
$n = 50$	3	10	78	10	77	12	93	12	94
Cal1s	1	11	85	10	81	14	109	12	98
Start 1	2	10	78	12	95	18	139	14	109
$n = 100$	3	10	79	11	89	15	120	16	125
Cal2s	1	7	50	6	43	9	64	11	71
Start 1	2	8	57	8	57	10	71	11	81
$n = 50$	3	7	50	7	50	10	71	9	68
Cal2s	1	7	50	7	51	11	78	12	88
Start 1	2	9	64	7	50	13	92	15	111
$n = 100$	3	8	57	6	44	11	78	10	73
Cal3s	1	7	50	9	64	10	71	8	60
Start 1	2	7	50	9	64	12	85	9	67
$n = 50$	3	7	50	11	78	11	78	9	69
Cal3s	1	7	50	13	92	13	92	10	76
Start 1	2	7	50	11	78	14	99	11	83
$n = 100$	3	7	50	16	113	14	99	11	81
QORs	1	6	55	6	55	6	55	7	64
Start 1	2	6	55	6	55	6	55	7	64
$n = 50$	3	6	55	6	55	6	55	7	65
GORs	1	7	64	8	73	8	74	9	84
Start 1	2	7	64	8	73	8	74	9	85
$n = 50$	3	7	65	8	73	8	73	9	83
ChaRs	1	14	74	14	74	15	77	16	79
Start 4	2	14	77	15	79	15	81	15	74
$n = 25$	3	11	56	12	58	12	58	16	81

Table 4E—Totals. To compute the sparse totals, non-sparse results were used whenever a sparse result was not available.

Function	PC	TN1		TN2		TN3		MINR	
Totals	1	222	1404	232	1473	243	1553	260	1613
sparse	2	227	1492	239	1567	262	1712	269	1695
	3	215	1341	236	1479	250	1620	266	1683
Totals	1	222	2709	232	2580	243	2553	260	2597
regular	2	227	3477	239	3417	262	3180	269	3276
	3	215	2539	236	2613	250	2581	266	4752

Tables 5—Comparison of various truncated-Newton routines against other optimization algorithms. All test functions are used with $\eta = .25, .1, .001$.

Table 5A—Smaller functions. Differencing along search direction.

Function	η	PLMA		QNM		MNA		TN		PBTN		BTN	
Pen1	.25	22	53	27	33	17	18	7	29	10	46	10	46
Start 3	.1	8	27	8	26	9	25	7	30	10	48	10	48
$n = 50$	.001	8	32	8	31	7	29	3	19	6	40	6	40
Pen2	.25	52	118	134	242	17	17	11	64	10	60	14	89
Start 3	.1	28	76	99	322	9	31	12	78	10	64	13	86
$n = 50$	.001	15	71	73	341	6	26	12	95	9	67	12	118
Pen3	.25	40	76	67	135	40	44	10	47	11	60	10	62
Start 3	.1	38	76	63	150	12	44	9	46	9	52	10	61
$n = 50$	.001	28	71	56	155	11	48	9	54	10	63	10	72
GenR	.25	108	201	128	287	62	202	31	330	42	584	33	499
Start 2	.1	119	263	118	323	66	257	33	348	36	469	32	518
$n = 50$	.001	119	330	118	412	88	392	34	395	37	468	35	614
Cal1	.25	194	366	162	191	7	9	10	199	8	227	16	978
Start 1	.1	204	401	89	214	6	11	9	158	8	232	17	1151
$n = 50$	.001	205	456	88	269	6	17	9	166	7	252	13	787
Cal2	.25	64	106	28	52	4	4	7	69	7	104	8	133
Start 1	.1	61	118	28	54	4	4	7	69	7	104	8	133
$n = 50$	.001	60	123	28	67	4	6	7	71	7	107	8	135
Cal3	.25	80	152	90	114	6	6	7	112	7	125	7	206
Start 1	.1	78	155	59	135	5	7	7	112	7	127	7	206
$n = 50$	.001	77	161	59	161	5	11	7	113	7	125	8	289

Table 5B—Larger functions. Differencing along search direction.

Function	η	PLMA		QNM	MNA	TN		PBTN		BTN	
Pen1	.25	17	40	NR	NR	2	11	2	11	2	11
Start 3	.1	2	9	NR	NR	2	11	2	11	2	11
$n = 100$	.001	2	10	NR	NR	2	12	2	12	2	12
Pen2	.25	14	28	NR	NR	6	29	6	26	6	25
Start 3	.1	7	18	NR	NR	6	30	5	23	5	24
$n = 100$	.001	7	29	NR	NR	6	41	5	33	5	36
Pen3	.25	49	85	NR	NR	11	65	11	75	13	79
Start 3	.1	48	94	NR	NR	11	67	11	75	13	89
$n = 100$	.001	35	83	NR	NR	11	77	11	89	11	91
GenR	.25	191	365	NR	NR	57	684	73	1055	60	1150
Start 2	.1	192	410	NR	NR	60	775	72	1012	62	1153
$n = 100$	.001	188	528	NR	NR	58	782	63	1062	60	1197
Cal1	.25	423	819	NR	NR	10	409	9	828	26	3855
Start 1	.1	429	854	NR	NR	10	409	8	570	23	3214
$n = 100$	.001	416	905	NR	NR	10	372	8	602	26	3948
Cal2	.25	112	204	NR	NR	8	117	7	172	8	256
Start 1	.1	107	206	NR	NR	8	117	7	172	8	256
$n = 100$	.001	113	228	NR	NR	8	122	8	199	8	259
Cal3	.25	143	270	NR	NR	7	159	7	196	9	508
Start 1	.1	142	281	NR	NR	7	168	7	196	9	636
$n = 100$	.001	138	284	NR	NR	7	176	7	195	9	617

Table 5C—Miscellaneous smaller functions. Differencing along search direction.

Function	η	PLMA		QNM		MNA		TN		PBTN		BTN	
Cheb	.25	38	75	32	65	29	121	7	53	10	87	10	104
Start 2	.1	33	71	28	67	24	116	8	68	9	105	9	96
$n = 20$	.001	33	90	28	92	30	161	9	90	10	129	11	164
QOR	.25	14	29	23	39	3	3	6	25	5	25	5	29
Start 1	.1	14	29	13	27	3	3	6	25	5	25	5	29
$n = 50$	.001	14	29	13	27	3	3	6	25	5	25	5	29
GOR	.25	14	71	29	59	5	5	7	60	7	77	7	74
Start 1	.1	14	76	29	59	5	5	7	61	7	77	7	74
$n = 50$	.001	42	97	29	72	5	7	7	67	7	85	6	61
ChaR	.25	40	82	48	97	15	28	11	77	11	121	10	88
Start 4	.1	37	76	46	122	16	48	11	75	11	91	10	94
$n = 25$	.001	43	119	47	164	12	47	14	126	1	4	11	104

Table 5D—Sparse finite-differencing.

Function	η	PLMA		QNM		MNA		TN		PBTN		BTN	
GenRs	.25	108	201	128	287	62	202	31	168	42	239	33	196
Start 2	.1	119	263	118	323	66	257	33	198	36	218	32	197
$n = 50$	.001	119	330	118	412	88	392	34	245	37	258	35	252
GenRs	.25	191	365	NR		NR		57	335	73	415	60	346
Start 2	.1	192	410	NR		NR		60	370	72	430	62	398
$n = 100$	.001	188	528	NR		NR		58	422	63	474	60	441
Cal1s	.25	194	366	162	191	7	9	10	78	8	63	16	120
Start 1	.1	204	401	89	214	6	11	9	73	8	66	17	130
$n = 50$	.001	205	456	88	269	6	17	9	88	7	72	13	117
Cal1s	.25	423	819	NR		NR		10	79	9	72	26	191
Start 1	.1	429	854	NR		NR		10	85	8	68	23	178
$n = 100$	.001	416	905	NR		NR		10	95	8	79	26	227
Cal2s	.25	64	106	28	52	4	4	7	50	7	50	8	57
Start 1	.1	61	118	28	54	4	4	7	50	7	50	8	57
$n = 50$	.001	60	123	28	67	4	6	7	52	7	51	8	59
Cal2s	.25	112	204	NR		NR		8	57	7	50	8	57
Start 1	.1	107	206	NR		NR		8	57	7	50	8	57
$n = 100$	.001	113	228	NR		NR		8	59	8	58	8	59
Cal3s	.25	80	152	90	114	6	6	7	50	7	50	7	50
Start 1	.1	78	155	59	135	5	7	7	50	7	52	7	50
$n = 50$	.001	77	161	59	161	5	11	7	59	7	58	8	68
Cal3s	.25	143	270	NR		NR		7	50	7	50	9	57
Start 1	.1	142	281	NR		NR		7	51	7	52	9	65
$n = 100$	.001	138	284	NR		NR		7	60	7	60	9	77
QORs	.25	14	29	23	39	3	3	6	55	5	46	5	46
Start 1	.1	14	29	13	27	3	3	6	55	5	46	5	46
$n = 50$	.001	14	29	13	27	3	3	6	55	5	46	5	46
GORs	.25	14	71	29	59	5	5	7	65	7	65	7	64
Start 1	.1	14	76	29	59	5	5	7	66	7	65	7	64
$n = 50$	.001	42	97	29	72	5	7	7	73	7	71	6	59
ChaRs	.25	40	82	48	97	15	28	11	56	11	56	10	51
Start 4	.1	37	76	46	122	16	48	11	58	11	60	10	52
$n = 25$	.001	43	119	47	164	12	47	14	98	1	3	11	70

Table 5E—Totals. To compute the sparse totals, non-sparse results were used whenever a sparse result was not available.

Function	PLMA		QNM	MNA	TN		PBTN		BTN	
Totals, sparse	4773	10026	NA	NA	654	4478	684	4719	749	5368
Totals, regular	4773	10026	NA	NA	654	7989	684	10889	749	24644

Tables 6—Comparison of various values of η (.25, .1, .001, .5, .7, .9) for the best truncated-Newton routine. All test functions are used.

Table 6A—Differencing along search direction.

Function	n	.25		.1		.001		.5		.7		.9	
Pen1	50	7	29	7	30	3	19	16	41	16	41	16	41
Pen1	100	2	11	2	11	2	12	16	42	16	42	16	42
Pen2	50	11	64	12	78	12	95	18	52	18	52	18	51
Pen2	100	6	29	6	30	6	41	10	26	10	26	10	26
Pen3	50	10	47	9	46	9	54	13	56	13	56	13	56
Pen3	100	11	65	11	67	11	77	14	69	14	69	14	69
GenR	50	31	330	33	348	34	395	35	420	33	336	34	360
GenR	100	57	684	60	755	58	782	60	844	60	802	59	688
Cal1	50	10	199	9	158	9	166	9	190	9	190	9	165
Cal1	100	10	409	10	409	10	372	10	402	11	444	11	480
Cal2	50	7	69	7	69	7	71	7	69	7	69	7	69
Cal2	100	8	117	8	117	8	122	8	117	8	117	8	117
Cal3	50	7	112	7	112	7	113	7	112	7	112	7	112
Cal3	100	7	159	7	168	7	176	7	159	7	159	7	159
Cheb	20	7	53	8	68	9	90	7	53	9	70	12	119
QOR	50	6	25	6	25	6	25	6	25	6	25	6	25
GOR	50	7	60	7	61	7	67	7	60	7	60	7	60
ChaR	25	11	77	11	75	14	126	14	82	14	82	14	82
Totals		215	2539	220	2627	219	2803	264	2819	265	2752	268	2718

Table 6B—Sparse finite-differencing.

Function	n	.25		.1		.001		.5		.7		.9	
GenRs	50	31	168	33	198	34	245	35	197	33	175	34	191
GenRs	100	57	335	60	370	58	422	60	344	60	348	59	327
Cal1s	50	10	78	9	73	9	88	9	65	9	65	9	65
Cal1s	100	10	79	10	85	10	95	10	73	11	81	11	80
Cal2s	50	7	50	7	50	7	52	7	50	7	50	7	50
Cal2s	100	8	57	8	57	8	59	8	57	8	57	8	57
Cal3s	50	7	50	7	50	7	59	7	50	7	50	7	50
Cal3s	100	7	50	7	51	7	60	7	50	7	50	7	50
QORs	50	6	55	6	55	6	55	6	55	6	55	6	55
GORs	50	7	65	7	66	7	73	7	65	7	65	7	65
ChaRs	25	11	56	11	58	14	98	14	66	14	66	14	66
Totals		161	1043	165	1113	167	1316	170	1072	169	1062	169	1056

Tables 7—Comparison of truncated-Newton and modified-Newton algorithms, ignoring function/gradient evaluations required to compute the matrix/vector products for TN.

Table 7A—Smaller functions.

Function	η	MNA		TN	
Pen1	.25	17	18	7	18
Start 3	.1	9	25	7	19
$n = 50$	.001	7	29	3	15
Pen2	.25	17	17	11	35
Start 3	.1	9	31	12	42
$n = 50$	.001	6	26	12	57
Pen3	.25	40	44	10	14
Start 3	.1	12	44	9	16
$n = 50$	.001	11	48	9	24
GenR	.25	62	202	31	75
Start 2	.1	66	257	33	99
$n = 50$	.001	88	392	34	143
Cal1	.25	7	9	10	18
Start 1	.1	6	11	9	19
$n = 50$	.001	6	17	9	34
Cal2	.25	4	4	7	8
Start 1	.1	4	4	7	8
$n = 50$	.001	4	6	7	10
Cal3	.25	6	6	7	8
Start 1	.1	5	7	7	8
$n = 50$	.001	5	11	7	17

Table 7B—Miscellaneous smaller functions and totals.

Function	η	MNA		TN	
Cheb	.25	29	121	7	16
Start 2	.1	24	116	8	17
$n = 20$	.001	30	161	9	29
QOR	.25	3	3	6	7
Start 1	.1	3	3	6	7
$n = 50$	.001	3	3	6	7
GOR	.25	5	5	7	9
Start 1	.1	5	5	7	10
$n = 50$	.001	5	7	7	17
ChaR	.25	15	28	11	23
Start 4	.1	16	48	11	25
$n = 25$	.001	12	47	14	56
Totals		541	1753	347	910

Bibliography

Brent, R.P. (1973), "Some efficient algorithms for solving systems of non-linear equations," *SIAM Num. Anal.*, **10**, pp. 327–344.

Broyden, C.G. (1971), "The convergence of an algorithm for solving sparse non-linear systems," *Math. Comp.*, **25**, pp. 285–294.

Bunch, J.R., and Parlett, B.N. (1971), "Direct methods for solving symmetric indefinite systems of linear equations," *SIAM Num. Anal.*, **8**, pp. 639–655.

Bunch, J.R., and Rose, D.J. (1976), *Sparse Matrix Computations*, Academic Press, New York.

Concus, P., Golub, G., and O'Leary, D.P. (1976), "A generalized conjugate-gradient method for the numerical solution of elliptic partial differential equations," in *Sparse Matrix Computations* (J. Bunch and D. Rose, ed.), pp. 309–332, Academic Press, New York.

Davidon, W. (1959), "Variable metric methods for minimization," A.E.C. Res. and Develop. Report ANL–5990, Argonne National Laboratory.

Dax, A., and Kaniel, S. (1977), "Pivoting techniques for symmetric Gaussian elimination," *Num. Math.*, **28**, pp. 221–241.

Dembo, R.S., Eisenstat, S.C., and Steihaug,T. (1980), "Inexact Newton methods," Tech. Report Series B: 47, School of Organization and Management, Yale University.

Dembo, R.S., and Steihaug, T. (1980), "Truncated-Newton methods for large-scale optimization," presented at the ORSA/TIMS Joint National Meeting in Washington, DC, May 1980.

Dennis, J.E., and Moré, J.J. (1977), "Quasi-Newton methods, motivation and theory," *SIAM Review*, **19**, pp. 46–89.

Fenelon, M. (1981), "Preconditioned conjugate-gradient-type methods for large-scale unconstrained optimization," Ph.D. thesis, Dept. of Operations Research, Stanford University.

Fletcher, R. (1965), "Function minimization without evaluating derivatives— a review," *Comput. J.*, **8**, pp. 33–41.

Fletcher, R., and Reeves, C.M. (1964), "Function minimization by conjugate gradients," *Comput. J.*, **7**, pp. 149–154.

Forsythe, G.E., and Straus, E.G. (1955), "On best conditioned matrices," *Proc. Amer. Math. Soc.*, **6**, pp. 340–345.

Gill, P.E., Golub, G., Murray, W., and Saunders, M.A. (1974), "Methods for modifying matrix factorizations," *Math. Comp.*, **28**, pp. 505–536.

Gill, P.E., and Murray, W. (1972), "Quasi-Newton methods for unconstrained optimization," *J. Inst. Maths. Applics.*, **9**, pp. 91–108.

Gill, P.E., and Murray, W. (1973a), "The numerical solution of a problem in the calculus of variations," in *Recent Mathematical Developments in Control* (D.J. Bell, ed.), pp. 97–122, Academic Press, London and New York.

Gill, P.E., and Murray, W. (1973b), "Quasi-Newton methods for linearly constrained optimization," Report NAC 32, National Physical Laboratory, England.

Gill, P.E., and Murray, W. (1974a), "Newton-type methods for unconstrained and linearly constrained optimization," *Math. Prog*, **17**, pp. 311–350.

Gill, P.E., and Murray, W. (1974b), "Safeguarded steplength algorithms for optimization using descent methods," Report NAC 37, National Physical Laboratory, England.

Gill, P.E., and Murray, W. (1976), "Nonlinear least squares and nonlinearly constrained optimization," in *Numerical Analysis, Lecture Notes in Mathematics no. 506* (G.A. Watson, ed.), pp. 134–147, Springer-Verlag, Berlin.

Gill, P.E., and Murray, W. (1978), "Algorithms for the solution of the nonlinear least-squares problem," *SIAM Num. Anal.*, **15**, pp. 977–992.

Gill, P.E., and Murray, W. (1979), "Conjugate-gradient methods for large-scale nonlinear optimization," Report SOL 79–15, Operations Research Dept., Stanford University.

Gill, P.E., Murray, W., and Pitfield, R.A. (1972), "The implementation of two revised quasi-Newton algorithms for unconstrained optimization," Report NAC 11, National Physical Laboratory, England.

Greenstadt, J.L. (1967), "On the relative inefficiencies of gradient methods," *Math. Comp*, **21**, pp. 360–367.

Hebden, M.D. (1973), "An algorithm for minimization using exact second derivatives," Tech. Report T.P. 515, A.E.R.E., Theoretical Physics Division, Harwell, England.

Hestenes, M. (1980), *Conjugate direction methods in optimization*, Springer-Verlag, Berlin.

Hestenes, M., and Stiefel, E. (1952), "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Standards*, **49**, pp. 409–436.

Lanczos, C. (1950), "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *J. Res. Nat. Bur. Standards*, **45**, pp. 255–282.

Luenberger, D.G. (1973), *Introduction to linear and nonlinear programming*, Addison-Wesley, Reading, MA.

Marwil, E.S. (1978), "Exploiting sparsity in Newton-like methods," Ph.D. thesis, Dept. of Computer Science, Cornell University.

McCormick, G.P., and Pearson, J.D. (1969), "Variable metric methods and unconstrained optimization," in *Optimization* (R. Fletcher, ed.), pp. 307–325, Academic Press, London and New York.

Murray, W. (1972), "Second derivative methods," in "Numerical methods for unconstrained optimization" (W. Murray, ed.), Academic Press, London and New York, pp. 57–71.

Murtagh, B.A. and Saunders, M.A. (1978), "Large-scale linearly constrained optimization," *Math. Prog.*, **14**, pp. 41–72.

Murtagh, B.A. and Saunders, M.A. (1980, revised Februrary 1981), "A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints," Report 80–1R, Operations Research Dept., Stanford University.

Ortega, J.M., and Rheinbolt, W.C. (1970), *Iterative solution of nonlinear equations in several variables*, Academic Press, London and New York.

Paige, C.C., and Saunders, M.A. (1975), "Solution of sparse indefinite systems of linear equations," *SIAM Num. Anal.*, **12**, pp. 617–629.

Parlett, B.N. (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ.

Parlett, B.N., and Scott, D.S. (1979), "The Lanczos algorithm with selective orthogonalization," *Math. Comp.*, **33**, pp. 217–238.

Powell, M.J.D. (1970), "A FORTRAN subroutine for unconstrained minimization, requiring first derivatives of the objective function," Report AERE-R 6469, Atomic Energy Research Establishment, Harwell, England.

Powell, M.J.D. (1976), "Some convergence properties of the conjugate gradient method," *Math. Prog.*, **11**, pp. 42–49.

Powell, M.J.D. (1977), "Restart procedures for the conjugate gradient method," *Math. Prog.*, **12**, pp. 241–254.

Powell, M.J.D., and Toint, P. (1979), "The estimation of sparse Hessian matrices," *SIAM Num. Anal.*, **16**, pp. 1060–1074.

Rosenbrock, H.H. (1960), "An automatic method for finding the greatest or least value of a function," *Comput. J.*, **3**, pp. 175–184.

Schubert, L.K. (1970), "Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian," *Math. Comp.*, **24**, pp. 27–30.

Sherman, A.H. (1978), "On Newton-iterative methods for the solution of systems of nonlinear equations," *SIAM Num. Anal.*, **15**, pp. 755–771.

Steihaug, T. (1980), "Quasi-Newton methods for large-scale nonlinear problems," Ph.D. thesis, School of Organization and Management, Yale University.

Stewart, G.W. (1967), "A modification of Davidon's method to accept difference approximations of derivatives," *J. ACM*, **14**, pp. 72–83.

Thapa, M. (1980), "Optimization of unconstrained functions with sparse Hessian matrices," Ph.D. thesis, Dept. of Operations Research, Stanford University.

Toint, P. (1978), "Some numerical results using a sparse matrix updating formula in unconstrained optimization," *Math. Comp.*, **32**, pp. 839–851.

van der Sluis, A. (1969), "Condition numbers and equilibration of matrices," *Num. Math.*, **14**, pp. 14–23.

Vardi, A. (1980), "A trust region algorithm for unconstrained minimization: convergence properties and implementation," ICASE Report 80–35.

Wilkinson, J.H. (1965), *The algebraic eigenvalue problem*, Oxford Univerity Press, London.